Remote ATtestation ProcedureS
Internet-Draft
Intended status: Standards Track
Expires: 11 June 2026

S. Frost
Arm
T. Fossati
Linaro
H. Tschofenig
H-BRS
H. Birkholz
Fraunhofer SIT

8 December 2025

# EAT Measured Component
# draft-ietf-rats-eat-measured-component-09

## Abstract

The term "measured component" refers to an object within the attester’s target environment whose state can be sampled and typically digested using a cryptographic hash function. Examples of measured components include firmware stored in flash memory, software loaded into memory at start time, data stored in a file system, or values in a CPU register. This document provides the information model for the "measured component" and two associated data models. This separation is intentional: the JSON and CBOR serializations, coupled with the media types and associated CoAP Content-Formats, enable the immediate use of the semantics within the EAT framework. Meanwhile, the information model can be reused in future specifications to provide additional serializations, for example using ASN.1.

## Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Remote ATtestation ProcedureS Working Group mailing list (rats@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/rats/.

Source for this draft and an issue tracker can be found at https://github.com/thomas-fossati/draft-fft-rats-eat-measured-component.

## 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 11 June 2026.

## Copyright Notice

Copyright (c) 2025 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

## 1. Introduction

Section 4.2.16 of [RFC9711] defines a Measurements claim that:

> "[c]ontains descriptions, lists, evidence or measurements of the software that exists on the entity or any other measurable subsystem of the entity."

This claim allows for different measurement formats, each identified by a different CoAP Content-Format (Section 12.3 of [RFC7252]). Currently, the only specified format is CoSWID of type "evidence", as per Section 2.9.4 of [RFC9393]. However, CoSWID is not suitable for measurements that cannot be anchored to a file system, such as those in early boot environments. To address this gap, this document introduces a "measured component" format that can be used with the EAT Measurements claim alongside or instead of CoSWID.

The term "measured component" refers to an object within the attester's target environment whose state can be sampled and typically digested using a cryptographic hash function. This includes, for example: the invariant part of a firmware component that is loaded in memory at startup time, a run-time integrity check (RTIC), a file system object, or a CPU register.

This document provides the information model for the "measured component" and two associated data models [RFC3444]. This separation is intentional: the JSON and CBOR serializations, coupled with the media types and associated CoAP Content-Formats, enable the immediate use of the semantics within the EAT framework. Meanwhile, the information model can be reused in future specifications to provide additional serializations, for example using ASN.1.

## 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.

In this document, CDDL [RFC8610] [RFC9165] [RFC9741] is used to describe the data formats.

## 3. Information Model

This section presents the information model of a "measured component".

A "measured component" information element includes the component's sampled state (in digested or raw form) along with metadata that helps in identifying the component. Optionally, any entities responsible for signing the installed component can also be specified.

The information elements (IE) that constitute a "measured component" are described in Table 1.

| IE | Description | Requirement Level |
|---|---|---|
| Component Name | The name given to the measured component. It is important that this name remains consistent across different releases to allow for better tracking of the same measured item across updates. When combined with a consistent versioning scheme, it enables better signalling from the appraisal procedure to the relying parties. | REQUIRED |
| Component Version | A value representing the specific release or development version of the measured component. Using Semantic Versioning (https://semver.org/spec/v2.0.0.html) is RECOMMENDED. | OPTIONAL |
| Digested or Raw Value | Either the raw value or the digested value of the measured component. | REQUIRED |
| Digest Algorithm | Hash algorithm used to compute the Digest Value. | REQUIRED only if the value is in the digested form |
| Authorities | One or more entities that can authoritatively identify the component being measured. | OPTIONAL |

Table 1: Measured Component Information Elements

A data model implementing this information model SHOULD also allow a limited amount of extensibility to accommodate profile-specific semantics.

## 4. Data Model

This section presents a JSON and CBOR data model that implements the information model outlined in Section 3.

The data model is inspired by the "PSA software component" claim (Section 4.4.1 of [RFC9783]), which has been refactored to take into account the recommendations about the design of new EAT claims described in Appendix E of [RFC9711].

CDDL is used to express rules and constraints of the data model for both JSON and CBOR. These rules must be strictly followed when creating or validating "measured component" data items. When there is variation between CBOR and JSON, the CDDL generic JC<>, defined in Appendix D of [RFC9711], is used.

## 4.1. Common Types

The following three basic types are used at various places within the measured component data model:

```
bytes-b64u = text .b64u bytes
bytes8 = bytes .size 8
bytes8-b64u = text .b64u bytes8
```

## 4.2. The digest Type

A digest represents the result of a hashing operation together with the hash algorithm used. The type of the digest algorithm identifier can be either int or text and is interpreted according to the [IANA.named-information] registry. Specifically, int values are matched against "ID" entries and text values are matched against "Hash Name String" entries. Whenever possible, using the int encoding is RECOMMENDED.

```
digest = [
  alg: (int / text)
  val: digest-value-type
]

digest-value-type = eat.JC<bytes-b64u, bytes>
```

## 4.3. The measured-component Data Item

The measured-component data item is as follows:

```
measured-component = {
  id-label => component-id
  measurement
  ? authorities-label => [ + authority-id-type ]
  ? flags-label => flags-type
}

measurement //= ( digested-measurement-label => digest )
measurement //= ( raw-measurement-label => bytes )

authority-id-type = eat.JC<bytes-b64u, bytes>
flags-type = eat.JC<bytes8-b64u, bytes8>

id-label = eat.JC<"id", 1>
digested-measurement-label = eat.JC<"digested-measurement", 2>
raw-measurement-label = eat.JC<"raw-measurement", 5>
authorities-label = eat.JC<"authorities", 3>
flags-label = eat.JC<"flags", 4>
```

The members of the measured-component CBOR map / JSON object are:

"id" (index 1):
: The measured component identifier encoded according to the format described in Section 4.3.1.

"measurement":
: Either a digest value and digest algorithm (index 2), encoded using the digest format (Section 4.2), or the "raw" measurement (index 5), encoded as a byte string. Note that, while the size of the digested form is constrained by the digest function, the size of the raw form can vary greatly depending on what is being measured (it could be a CPU register or an entire configuration blob, for example). Therefore, a decoder implementation may decide to limit the amount of memory it allocates to this specific field.

"authorities" (index 3):
: One or more authorities, see Section 4.3.2.

"flags" (index 4):
: a 64-bit field with profile-defined semantics, see Section 4.3.3.

#### 4.3.1. Component Identifier

The component-id data item is as follows:

```
component-id = [
  name:      text
  ? version: version
]

;# import coswid.$version-scheme from rfc9393 as coswid

version = [
  val:      text
  ? scheme: coswid.$version-scheme
]
```

name A string that provides a human readable identifier for the component in question. Format and adopted conventions depend on the component type.

version A compound version data item that reuses encoding and semantics of [RFC9711] sw-version-type.

#### 4.3.2. Authority

An authority is an entity that can authoritatively identify a given component by digitally signing it. This signature is usually verified during installation, or when the measured component is executed by the boot ROM, operating system, or application launcher. For example, as in UEFI Secure Boot [UEFI2] and Arm Trusted Board Boot [TBBR-CLIENT]. Another example may be the controlling entity in an app store.

An authority is identified by its signing key. It could be an X.509 certificate, a raw public key, a public key thumbprint, or some other identifier that can be uniquely associated with the signing entity. In some cases, multiple parties may need to sign a component to indicate their endorsement or approval. This could include roles such as a firmware update system, fleet owner, or third-party auditor. The specific purpose of each signature may depend on the deployment, and the order of authorities within the array could indicate meaning.

If an EAT profile (Section 6 of [RFC9711]) uses measured components, it MUST specify whether the authorities field is used. If it is used, the profile MUST also specify what each of the entries in the authorities array represents, and how to interpret the corresponding authority-id-type.

The authority-id-type is defined as follows:

```
authority-id-type = eat.JC<bytes-b64u, bytes>
```

#### 4.3.3. Profile-specific Flags

This optional field can contain up to 64 bits of profile-defined semantics, enabling a profile of this specification to encode additional information and extend the base type. It can be used to carry information in fixed-size chunks, such as a bit mask or a single value within a predetermined set of codepoints. Regardless of its internal structure, the size of this field is exactly 8 bytes.

The flags-type is defined as follows:

```
flags-type = eat.JC<bytes8-b64u, bytes8>
```

If an EAT profile (Section 6 of [RFC9711]) uses measured components, it MUST specify whether the flags field is used. If it is used, the profile MUST also specify how to interpret the 64 bits.

### 4.4. EAT measurements-format Extensions

The CDDL in Figure 1 extends the $measurements-body-cbor and $measurements-body-json EAT sockets to add support for measured-components to the Measurements claim.

```
mc-cbor = bytes .cbor measured-component
mc-json = text .json measured-component

; EAT CBOR (‘.feature "cbor"‘)
$measurements-body-cbor /= mc-cbor ; native
$measurements-body-cbor /= mc-json ; tunnel

; EAT JSON (‘.feature "json"‘)
$measurements-body-json /= mc-json            ; native
$measurements-body-json /= text .b64u mc-cbor ; tunnel
```

Figure 1: EAT measurements-format Extensions

Each socket is extended with two new types: a "native" representation that is used when measured-component and the EAT have the same serialization (e.g., they are both CBOR), and a "tunnel" representation that is used when the serializations differ.

### 4.5. measurements-format for CBOR EAT

The entries in Table 2 are the allowed content-type / content-format pairs when the measured-component is carried in a CBOR EAT.

Note the use of the "native" and "tunnel" formats from Figure 1, and how the associated CoAP Content-Format is used to describe the original serialization.

| content-type (CoAP C-F equivalent) | content-format |
|---|---|
| application/measured-component+cbor | mc-cbor |
| application/measured-component+json | mc-json |

Table 2: measurement-format for EAT CWT

## 4.6. measurements-format for JSON EAT

Table 3 is the equivalent of Table 2 for JSON-serialized EAT.

| content-type (CoAP C-F equivalent) | content-format |
|---|---|
| application/measured-component+json | mc-json |
| application/measured-component+cbor | tstr .b64u mc-cbor |

Table 3: measurement-format for EAT JWT

## 4.7. EAT Profiles and Measured Components

The semantics of the authorities and profile flags fields are defined by the applicable EAT profile, i.e., the profile of the wrapping EAT.

If the profile of the EAT is not known to the consumer and one or more Measured Components within that EAT include authorities and/or profile flags, the consumer MUST reject the EAT.

## 4.8. Examples

The example in Figure 2 is a digested measured component with all the fields populated.

```
{
  / id / 1: [
    / name / "boot loader X",
    / version / [
      "1.2.3rc2",
      16384 / semver /
    ]
  ],
  / measurement / 2: [
    / alg / "sha-256",
    / val / h'3996003d486fb91ffb056f7d03f2b2992b215b31dbe7af4b37
              3431fc7d319da3'
  ],
  / authorities / 3: [
    h'492e9b676c21f6012b1ceeb9032feb4141a880797355f6675015ec59c5
      1ca1ec',
    h'4277bb97ba7b51577a0d38151d3e08b40bdf946753f5b5bdeb814d6ff5
      7a8a5e'
  ],
  / flags / 4: h'0000000000000101'
}
```

Figure 2: Complete Measured Component

The example in Figure 3 is the same measured component as above but used as the format of a measurements claim in a EAT claims-set.

This example uses TBD1 as the content-type value of the measurements-format entry. (This will change to the value assigned by IANA to the mc+cbor Content-Format.)

Note that the array contains only one measured component, but additional entries could be added if the measured TCB is made of multiple, individually measured components.

```
{
  273: [
    [
      TBD1, / mc+cbor /
      <<
        {
          / id / 1: [
            / name / "boot loader X",
            / version / [
              "1.2.3rc2",
              16384 / semver /
            ]
          ],
          / measurement / 2: [
            / alg / "sha-256",
            / val / h'3996003d486fb91ffb056f7d03f2b2992b215b31db
                      e7af4b373431fc7d319da3'
          ],
          / authorities / 3: [
            h'492e9b676c21f6012b1ceeb9032feb4141a880797355f66750
              15ec59c51ca1ec',
            h'4277bb97ba7b51577a0d38151d3e08b40bdf946753f5b5bdeb
              814d6ff57a8a5e'
          ]
        }
      >>
    ]
  ]
}
```

Figure 3: EAT Measurements Claim using a Measured Component (CBOR)

The example in Figure 4 illustrates the inclusion of a JSON measured component inside a JSON EAT.

This example uses TBD2 as the content-type value of the measurements-format entry. (This will change to the value assigned by IANA to the mc+json Content-Format.)

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

{
  "measurements": [
    [
      TBD2, / mc+json /
      "{ \"id\": [ \"boot loader X\", [ \"1.2.3rc2\", 16384 ] ], \"\
digested-measurement\": [ \"sha-256\", \"\
OZYAPUhvuR_7BW99A_KymSshWzHb569LNzQx_H0xnaM\" ], \"authorities\": [ \
\"SS6bZ2wh9gErHO65Ay_rQUGogHlzVfZnUBXsWcUcoew\", \"\
                  Qne7l7p7UVd6DTgVHT4ItAvflGdT9bW964FNb_V6il4\" ] }"
    ]
  ]
}
```

Figure 4: EAT Measurements Claim using a Measured Component (JSON)

The example in Figure 5 is a measured component representing a boot loader identified by its path name:

```
{
  / id / 1: [
    / name / "/boot/loader.bin"
  ],
  / measurement / 2: [
    / alg / "sha-384",
    / val / h'66ec2fb4e02d8c8b3eee320e750d9389d66c52c51db11cc6
              9cc5e410816283ed60ba573795f5fcc85e513af57b3f6def'
  ],
  / flags / 4: h'0000000000000101'
}
```

Figure 5: Digested Measured Component using File Path as Identifier

The example in Figure 6 is a raw measured component.

```
{
  / id / 1: [
    / name / "hardware-config"
  ],
  / measurement / 5: h'4f6d616861'
}
```

Figure 6: Raw Measured Component

## 5. Security Considerations

Please review Sections 9.1 (Claim Trustworthiness), 9.4 (Multiple EAT Consumers) and 9.5 (Detached EAT Bundle Digest Security Considerations) of [RFC9711]; these considerations apply to this document as well. Note that similar security considerations may apply when the Measured Component information model is serialized using different data models than the ones specified in this document.

The Component Name and Component Version can give an attacker detailed information about the software running on a device and its configuration settings. This information could offer an attacker valuable insights.

Any textual fields (e.g., Component Name and Component Version) that are stored in a file, inserted into a database, or displayed to humans must be properly sanitized to prevent attacks and undesirable behavior. Further discussion and references on this topic can be found in Section 7 of [RFC9839].

If the component measurement is digested, the digest must be computed using a strong cryptographic hash function.

## 6. Privacy Considerations

Please review Section 9.1 (Multiple EAT Consumers) of [RFC9711]; the differential encryption considerations discussed there also apply to this document.

The Component Name and Component Version could reveal private information about a device and its owner.

Additionally, the stability requirement of the Component Name could enable tracking.

## 7. IANA Considerations

// RFC Editor: replace "RFCthis" with the RFC number assigned to this document.

### 7.1. Media Types Registrations

IANA is requested to add the following media types to the "Media Types" registry [IANA.media-types].

| Name | Template | Reference |
|---|---|---|
| mc+cbor | application/measured-component+cbor | RFCthis |
| mc+json | application/measured-component+json | RFCthis |

Table 4: Measured Component Media Types

#### 7.1.1. application/measured-component+cbor

Type name: application
Subtype name: measured-component+cbor
Required parameters: n/a
Optional parameters: n/a
Encoding considerations: binary (CBOR)
Security considerations: Section 5 of RFCthis
Interoperability considerations: n/a
Published specification: RFCthis
Applications that use this media type: Attesters, Verifiers and Relying Parties
Fragment identifier considerations: The syntax and semantics of fragment identifiers are as specified for "application/cbor". (No fragment identification syntax is currently defined for "application/cbor".)
Person & email address to contact for further information: RATS WG mailing list (rats@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF
Provisional registration: no

#### 7.1.2. application/measured-component+json

Type name: application
Subtype name: measured-component+json
Required parameters: n/a
Optional parameters: n/a
Encoding considerations: binary (JSON is UTF-8-encoded text)
Security considerations: Section 5 of RFCthis
Interoperability considerations: n/a
Published specification: RFCthis
Applications that use this media type: Attesters, Verifiers and Relying Parties
Fragment identifier considerations: The syntax and semantics of

fragment identifiers are as specified for "application/json". (No fragment identification syntax is currently defined for "application/json".)
Person & email address to contact for further information: RATS WG mailing list (rats@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF
Provisional registration: no

### 7.2. Measured Component Content-Format Registrations

IANA is requested to register two Content-Format numbers in the "CoAP Content-Formats" sub-registry, within the "Constrained RESTful Environments (CoRE) Parameters" Registry [IANA.core-parameters], as follows:

| Content-Type | Content Coding | ID | Reference |
|---|---|---|---|
| application/measured-component+cbor | - | TBD1 | RFCthis |
| application/measured-component+json | - | TBD2 | RFCthis |

Table 5

If possible, TBD1 and TBD2 should be assigned in the 256..9999 range.

## 8. References

### 8.1. Normative References

[IANA.core-parameters]
IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters>.

[IANA.media-types]
IANA, "Media Types", <https://www.iana.org/assignments/media-types>.

[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>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/rfc/rfc7252>.

[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>.

[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>.

[RFC9165] Bormann, C., "Additional Control Operators for the Concise Data Definition Language (CDDL)", RFC 9165, DOI 10.17487/RFC9165, December 2021, <https://www.rfc-editor.org/rfc/rfc9165>.

[RFC9711] Lundblade, L., Mandyam, G., O'Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", RFC 9711, DOI 10.17487/RFC9711, April 2025, <https://www.rfc-editor.org/rfc/rfc9711>.

[RFC9741] Bormann, C., "Concise Data Definition Language (CDDL): Additional Control Operators for the Conversion and Processing of Text", RFC 9741, DOI 10.17487/RFC9741, March 2025, <https://www.rfc-editor.org/rfc/rfc9741>.

## 8.2. Informative References

[RFC3444] Pras, A. and J. Schoenwaelder, "On the Difference between Information Models and Data Models", RFC 3444, DOI 10.17487/RFC3444, January 2003, <https://www.rfc-editor.org/rfc/rfc3444>.

[RFC9393] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", RFC 9393, DOI 10.17487/RFC9393, June 2023, <https://www.rfc-editor.org/rfc/rfc9393>.

[RFC9783] Tschofenig, H., Frost, S., Brossard, M., Shaw, A., and T. Fossati, "Arm's Platform Security Architecture (PSA) Attestation Token", RFC 9783, DOI 10.17487/RFC9783, June 2025, <https://www.rfc-editor.org/rfc/rfc9783>.

[RFC9839] Bray, T. and P. Hoffman, "Unicode Character Repertoire Subsets", RFC 9839, DOI 10.17487/RFC9839, August 2025, <https://www.rfc-editor.org/rfc/rfc9839>.

[TBBR-CLIENT] Arm Ltd, "Trusted Board Boot Requirements Client (TBBR-CLIENT) Armv8-A", ARM DEN0006D, September 2018, <https://developer.arm.com/documentation/den0006>.

[UEFI2] UEFI Forum, Inc., "Unified Extensible Firmware Interface (UEFI) Specification", August 2022, <https://uefi.org/sites/default/files/resources/UEFI_Spec_2_10_Aug29.pdf>.

## Appendix A. Collected CDDL

This appendix contains all the CDDL definitions included in this specification.

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

measured-component = {
  id-label => component-id,
  measurement,
  ? authorities-label => [+ authority-id-type],
  ? flags-label => flags-type,
}
measurement //= (digested-measurement-label => digest // raw-\
                                              measurement-label => bytes)
authority-id-type = eat.JC<bytes-b64u, bytes>
flags-type = eat.JC<bytes8-b64u, bytes8>
component-id = [
  name: text,
  ? version: version,
]
version = [
  val: text,
  ? scheme: coswid.$version-scheme,
]
digest = [
  alg: int / text,
  val: digest-value-type,
]
```

```
digest-value-type = eat.JC<bytes-b64u, bytes>
bytes-b64u = text .b64u bytes
bytes8 = bytes .size 8
bytes8-b64u = text .b64u bytes8
id-label = eat.JC<"id", 1>
digested-measurement-label = eat.JC<"digested-measurement", 2>
raw-measurement-label = eat.JC<"raw-measurement", 5>
authorities-label = eat.JC<"authorities", 3>
flags-label = eat.JC<"flags", 4>
mc-cbor = bytes .cbor measured-component
mc-json = text .json measured-component
$measurements-body-cbor /= mc-cbor / mc-json
$measurements-body-json /= mc-json / text .b64u mc-cbor
eat.JSON-ONLY<J> = J .feature "json"
eat.CBOR-ONLY<C> = C .feature "cbor"
eat.JC<J, C> = eat.JSON-ONLY<J> / eat.CBOR-ONLY<C>
coswid.$version-scheme /= coswid.multipartnumeric / coswid.\
multipartnumeric-suffix / coswid.alphanumeric / coswid.decimal / \
                                          coswid.semver / int / text
coswid.multipartnumeric = 1
coswid.multipartnumeric-suffix = 2
coswid.alphanumeric = 3
coswid.decimal = 4
coswid.semver = 16384
```

## Appendix B. Open Issues

The list of currently open issues for this documents can be found at https://github.com/ietf-rats-wg/draft-ietf-rats-eat-measured-component/issues.

// Note to RFC Editor: please remove before publication.

## Acknowledgments

The authors would like to thank Carl Wallace, Carsten Bormann, Dionna Glaze, Giridhar Mandyam, Houda Labiod, Ionu Mihalcea, Jun Zhang, Laurence Lundblade, Michael Richardson, Muhammad Usama Sardar and Yogesh Deshpande for providing comments, reviews and suggestions that greatly improved this document.

The authors would also like to thank Ken Takayama for providing an implementation of this specification in the veraison/eat package.

## Authors’ Addresses

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

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

Hannes Tschofenig
University of Applied Sciences Bonn-Rhein-Sieg
Email: Hannes.Tschofenig@gmx.net

Henk Birkholz
Fraunhofer SIT
Email: henk.birkholz@ietf.contact