TEEP
Internet-Draft
Intended status: Standards Track
Expires: 4 September 2025

H. Tschofenig
M. Pei
Broadcom
D. Wheeler
Amazon
D. Thaler
Microsoft
A. Tsukamoto
Openchip & Software Technologies, S.L.

3 March 2025

Trusted Execution Environment Provisioning (TEEP) Protocol
draft-ietf-teep-protocol-21

Abstract

This document specifies a protocol that installs, updates, and deletes Trusted Components in a device with a Trusted Execution Environment (TEE). This specification defines an interoperable protocol for managing the lifecycle of Trusted Components.

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 4 September 2025.

Copyright Notice

Table of Contents

1. Introduction

The Trusted Execution Environment (TEE) concept has been designed to separate a regular operating system, also referred as a Rich Execution Environment (REE), from security-sensitive applications. In a TEE ecosystem, device vendors may use different operating systems in the REE and may use different types of TEEs. When Trusted Component Developers or Device Administrators use Trusted Application Managers (TAMs) to install, update, and delete Trusted Applications and their dependencies on a wide range of devices with potentially different TEEs then an interoperability need arises.

This document specifies the protocol for communicating between a TAM and a TEEP Agent.

The Trusted Execution Environment Provisioning (TEEP) architecture document [RFC9397] provides design guidance and introduces the necessary terminology.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification re-uses the terminology defined in [RFC9397].

As explained in Section 4.4 of that document, the TEEP protocol treats each Trusted Application (TA), any dependencies the TA has, and personalization data as separate components that are expressed in SUIT manifests, and a SUIT manifest might contain or reference multiple binaries (see [I-D.ietf-suit-manifest] for more details).

As such, the term Trusted Component (TC) in this document refers to a set of binaries expressed in a SUIT manifest, to be installed in a TEE. Note that a Trusted Component may include one or more TAs and/or configuration data and keys needed by a TA to operate correctly.

Each Trusted Component is uniquely identified by a SUIT Component Identifier (see [I-D.ietf-suit-manifest] Section 8.7.2.2).

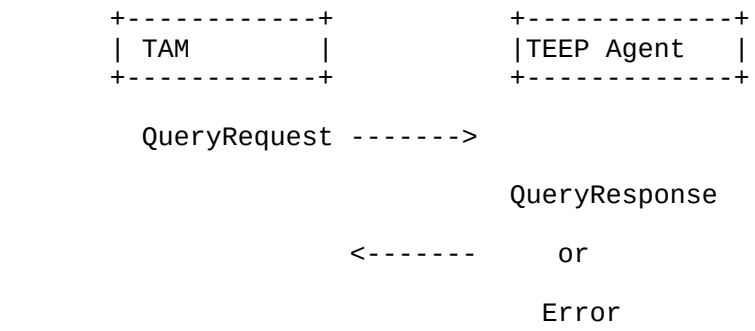
Attestation related terms, such as Evidence and Attestation Results, are as defined in [RFC9334].

3. Message Overview

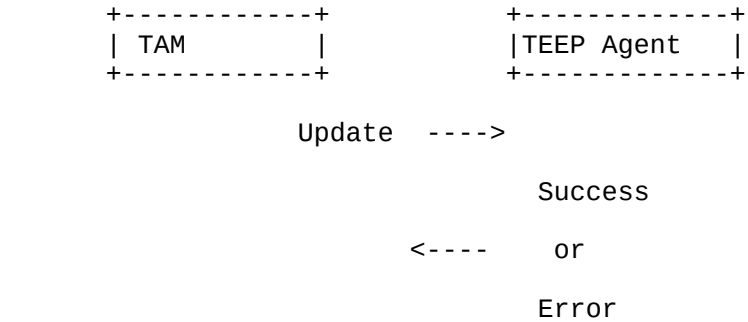
The TEEP protocol consists of messages exchanged between a TAM and a TEEP Agent. The messages are encoded in CBOR and designed to provide end-to-end security. TEEP protocol messages are signed by the endpoints, i.e., the TAM and the TEEP Agent, but Trusted Applications may also be encrypted and signed by a Trusted Component Developer or Device Administrator. The TEEP protocol not only uses CBOR but also the respective security wrapper, namely COSE [RFC9052]. Furthermore, for software updates the SUIT manifest format [I-D.ietf-suit-manifest] is used, and for attestation the Entity Attestation Token (EAT) [I-D.ietf-rats-eat] format is supported although other attestation formats are also permitted.

This specification defines five messages: QueryRequest, QueryResponse, Update, Success, and Error.

A TAM queries a device's current state with a QueryRequest message. A TEEP Agent will, after authenticating and authorizing the request, report attestation information, list all Trusted Components, and provide information about supported algorithms and extensions in a QueryResponse message. An error message is returned if the request could not be processed. A TAM will process the QueryResponse message and determine whether to initiate subsequent message exchanges to install, update, or delete Trusted Applications.



With the Update message a TAM can instruct a TEEP Agent to install and/or delete one or more Trusted Components. The TEEP Agent will process the message, determine whether the TAM is authorized and whether the Trusted Component has been signed by an authorized Trusted Component Signer. A Success message is returned when the operation has been completed successfully, or an Error message otherwise.



4. Detailed Messages Specification

TEEP messages are protected by the COSE_Sign1 or COSE_Sign structure as described in Section 8.1. The TEEP protocol messages are described in CDDL format [RFC8610] below.

```
teep-message = $teep-message-type .within teep-message-framework

teep-message-framework = [
  type: $teep-type / $teep-type-extension,
  options: { * teep-option },
  * any; further elements, e.g., for data-item-requested
]

teep-option = (uint => any)

; messages defined below:
$teep-message-type /= query-request
$teep-message-type /= query-response
$teep-message-type /= update
$teep-message-type /= teep-success
$teep-message-type /= teep-error

; message type numbers, in one byte which could take a number from 0 to 23
$teep-type = (0..23)
TEEP-TYPE-query-request = 1
TEEP-TYPE-query-response = 2
TEEP-TYPE-update = 3
TEEP-TYPE-teep-success = 5
TEEP-TYPE-teep-error = 6
```

4.1. Creating and Validating TEEP Messages

4.1.1. Creating a TEEP message

To create a TEEP message, the following steps are performed.

1. Create a TEEP message according to the description below and populate it with the respective content. TEEP messages sent by TAMs (QueryRequest and Update) can include a "token". The TAM can decide, in any implementation-specific way, whether to include a token in a message. The first usage of a token generated by a TAM MUST be randomly created. Subsequent token values MUST be different for each subsequent message created by a TAM.

2. Create a COSE Header containing the desired set of Header Parameters. The COSE Header MUST be valid per the [RFC9052] specification.

3. Create a COSE_Sign1 or COSE_Sign object using the TEEP message as the COSE_Sign1 or COSE_Sign Payload; all steps specified in [RFC9052] for creating a COSE_Sign1 or COSE_Sign object MUST be followed.

4.1.2. Validating a TEEP Message

When a TEEP message is received (see the ProcessTeepMessage conceptual API defined in Section 6.2.1 of [RFC9397]), the following validation steps are performed. If any of the listed steps fail, then the TEEP message MUST be rejected.

1. Verify that the received message is a valid CBOR object.

2. Verify that the message contains a COSE_Sign1 or COSE_Sign structure.

3. Verify that the resulting COSE Header includes only parameters and values whose syntax and semantics are both understood and supported or that are specified as being ignored when not understood.

4. Follow the steps specified in Section 4 of [RFC9052] ("Signing Objects") for validating a COSE_Sign1 or COSE_Sign object. The COSE_Sign1 or COSE_Sign payload is the content of the TEEP message.

5. Verify that the TEEP message is a valid CBOR map and verify the fields of the TEEP message according to this specification.

4.2. QueryRequest Message

A QueryRequest message is used by the TAM to learn information from the TEEP Agent, such as the features supported by the TEEP Agent, including cipher suites and protocol versions. Additionally, the TAM can selectively request data items from the TEEP Agent by using the data-item-requested parameter. Currently, the following features are supported:

* Request for attestation information of the TEEP Agent,
* Listing supported extensions,
* Querying installed Trusted Components, and
* Request for logging information in SUIT Reports.

Like other TEEP messages, the QueryRequest message is signed, and the relevant CDDL snippet is shown below. The complete CDDL structure is shown in Appendix C.

```
query-request = [
  type: TEEP-TYPE-query-request,
  options: {
    ? token => bstr .size (8..64),
    ? supported-freshness-mechanisms => [ + $freshness-mechanism ],
    ? challenge => bstr .size (8..512),
    ? versions => [ + version ],
    ? attestation-payload-format => text,
    ? attestation-payload => bstr,
    ? suit-reports => [ + bstr ],
    * $$query-request-extensions,
    * $$teep-option-extensions
  },
  supported-teep-cipher-suites: [ + $teep-cipher-suite ],
  supported-suit-cose-profiles: [ + $suit-cose-profile ],
  data-item-requested: uint .bits data-item-requested
]

version = uint .size 4
ext-info = uint .size 4

; data items as bitmaps
data-item-requested = &(
  attestation: 0,
  trusted-components: 1,
  extensions: 2,
  suit-reports: 3,
)
```

The message has the following fields:

type
: The value of (1) corresponds to a QueryRequest message sent from the TAM to the TEEP Agent.

token
: The value in the token parameter is used to match responses to requests, such as to look up any implementation-specific state it might have saved about that request, or to ignore responses to older QueryRequest messages before some configuration changes were made that affected their content. This is particularly useful when a TAM issues multiple concurrent requests to a TEEP Agent. The token MUST be present if and only if the attestation bit is clear in the data-item-requested value. When the attestation bit is clear then a challenge will be included, which offers replay protection capabilities. The size of the token is at least 8 bytes (64 bits) and maximum of 64 bytes. The first usage of a token generated by a TAM MUST be randomly created. Subsequent

token values MUST be different for each request message to distinguish the correct response from multiple requests. The token value MUST NOT be used for other purposes, such as a TAM to identify the devices and/or a device to identify TAMs or Trusted Components. The TAM SHOULD set an expiration time for each token and MUST ignore any messages with expired tokens. The TAM MUST expire the token value after receiving the first response containing the token value and ignore any subsequent messages that have the same token value.

supported-teep-cipher-suites
: The supported-teep-cipher-suites parameter lists the TEEP cipher suites supported by the TAM. Details about the cipher suite encoding can be found in Section 8.1.

supported-suit-cose-profiles
: The supported-suit-cose-profiles parameter lists the SUIT profiles supported by the TAM for parsing SUIT Reports. Details about the cipher suite encoding can be found in Section 8.2.

data-item-requested
: The data-item-requested parameter indicates what information the TAM requests from the TEEP Agent in the form of a bitmap.

  attestation (1) With this value the TAM requests the TEEP Agent to return an attestation payload, whether Evidence (e.g., an EAT) or an Attestation Result, in the response.

  trusted-components (2) With this value the TAM queries the TEEP Agent for all installed Trusted Components.

  extensions (4) With this value the TAM queries the TEEP Agent for supported capabilities and extensions, which allows a TAM to discover the capabilities of a TEEP Agent implementation.

  suit-reports (8) With this value the TAM requests the TEEP Agent to return SUIT Reports in the response.

  Further values may be added in the future.

supported-freshness-mechanisms
: The supported-freshness-mechanisms parameter lists the freshness mechanism(s) supported by the TAM. Details about the encoding can be found in Section 9. If this parameter is absent, it means only the nonce mechanism is supported. It MUST be absent if the attestation bit is clear.

challenge

The challenge field is an optional parameter used for ensuring the freshness of attestation Evidence returned with a QueryResponse message. It MUST be absent if the attestation bit is clear or the Passport model is used. When a challenge is provided in the QueryRequest and Evidence in the form of an EAT is returned with a QueryResponse message then the challenge contained in the QueryRequest MUST be used to generate the EAT, by copying the challenge into the eat_nonce claim (Section 4.1 of Section 5) if the nonce-based freshness mechanism is used for attestation Evidence. For more details about freshness of Evidence see Section 9.

If any format other than EAT is used, it is up to that format to define the use of the challenge field.

versions
The versions parameter enumerates the TEEP protocol version(s) supported by the TAM. A value of 0 refers to the current version of the TEEP protocol. If this field is not present, it is to be treated the same as if it contained only version 0.

attestation-payload-format
The attestation-payload-format parameter indicates the IANA Media Type of the attestation-payload parameter, where media type parameters are permitted after the media type. For protocol version 0, the absence of this parameter indicates that the format is "application/eat+cwt; eat_profile=urn:ietf:rfc:rfcXXXX" (see [I-D.ietf-rats-eat-media-type] for further discussion). (RFC-editor: upon RFC publication, replace XXXX above with the RFC number of this document.) It MUST be present if the attestation-payload parameter is present and the format is not an EAT in CWT format with the profile defined below in Section 5.

attestation-payload
The attestation-payload parameter contains Evidence or an Attestation Result for the TEEP Agent to use to perform attestation of the TAM. If the attestation-payload-format parameter is absent, the attestation payload contained in this parameter MUST be an Entity Attestation Token following the encoding defined in [I-D.ietf-rats-eat]. See Section 4.3.1 for further discussion.

suit-reports
If present, the suit-reports parameter contains a set of "boot" (including starting an executable in an OS context) time SUIT Reports of the TAM as defined by SUIT_Report in Section 4 of [I-D.ietf-suit-report], encoded using COSE as discussed in Section 8.2. SUIT Reports can be useful in QueryRequest messages

to pass additional information about the TAM to the TEEP Agent without depending on a Verifier including the relevant information in the TAM's Attestation Results.

4.3. QueryResponse Message

The QueryResponse message is the successful response by the TEEP Agent after receiving a QueryRequest message. As discussed in Section 7.2, it can also be sent unsolicited if the contents of the QueryRequest are already known and do not vary per message.

Like other TEEP messages, the QueryResponse message is signed, and the relevant CDDL snippet is shown below. The complete CDDL structure is shown in Appendix C.

```
query-response = [
  type: TEEP-TYPE-query-response,
  options: {
    ? token => bstr .size (8..64),
    ? selected-version => version,
    ? attestation-payload-format => text,
    ? attestation-payload => bstr,
    ? suit-reports => [ + bstr ],
    ? tc-list => [ + system-property-claims ],
    ? requested-tc-list => [ + requested-tc-info ],
    ? unneeded-manifest-list => [ + SUIT_Component_Identifier ],
    ? ext-list => [ + ext-info ],
    * $$query-response-extensions,
    * $$teep-option-extensions
  }
]

requested-tc-info = {
  component-id => SUIT_Component_Identifier,
  ? tc-manifest-sequence-number => uint .size 8,
  ? have-binary => bool
}
```

The QueryResponse message has the following fields:

type
: The value of (2) corresponds to a QueryResponse message sent from the TEEP Agent to the TAM.

token

The value in the token parameter is used to match responses to requests. The value MUST correspond to the value received with the QueryRequest message if one was present, and MUST be absent if no token was present in the QueryRequest.

selected-version
The selected-version parameter indicates the TEEP protocol version selected by the TEEP Agent. The absence of this parameter indicates the same as if it was present with a value of 0.

attestation-payload-format
The attestation-payload-format parameter indicates the IANA Media Type of the attestation-payload parameter, where media type parameters are permitted after the media type. For protocol version 0, the absence of this parameter indicates that the format is "application/eat+cwt; eat_profile=urn:ietf:rfc:rfcXXXX" (see [I-D.ietf-rats-eat-media-type] for further discussion). (RFC-editor: upon RFC publication, replace XXXX above with the RFC number of this document.) It MUST be present if the attestation-payload parameter is present and the format is not an EAT in CWT format with the profile defined below in Section 5.

attestation-payload
The attestation-payload parameter contains Evidence or an Attestation Result. This parameter MUST be present if the QueryResponse is sent in response to a QueryRequest with the attestation bit set. If the attestation-payload-format parameter is absent, the attestation payload contained in this parameter MUST be an Entity Attestation Token following the encoding defined in [I-D.ietf-rats-eat]. See Section 4.3.1 for further discussion.

suit-reports
If present, the suit-reports parameter contains a set of "boot" (including starting an executable in an OS context) time SUIT Reports as defined by SUIT_Report in Section 4 of [I-D.ietf-suit-report], encoded using COSE as discussed in Section 8.2. If a token parameter was present in the QueryRequest message the QueryResponse message is in response to, the suit-report-nonce field MUST be present in the SUIT Report with a value matching the token parameter in the QueryRequest message. SUIT Reports can be useful in QueryResponse messages to pass information to the TAM without depending on a Verifier including the relevant information in Attestation Results.

tc-list
The tc-list parameter enumerates the Trusted Components installed on the device in the form of system-property-claims objects, as defined in Section 4 of [I-D.ietf-suit-report]. The system-

property-claims can be used to learn device identifying information and TEE identifying information for distinguishing which Trusted Components to install in the TEE. This parameter MUST be present if the QueryResponse is sent in response to a QueryRequest with the trusted-components bit set.

requested-tc-list
: The requested-tc-list parameter enumerates the Trusted Components that are not currently installed in the TEE, but which are requested to be installed, for example by an installer of an Untrusted Application that has a TA as a dependency, or by a Trusted Application that has another Trusted Component as a dependency. Requested Trusted Components are expressed in the form of requested-tc-info objects. A TEEP Agent can get this information from the RequestTA conceptual API defined in [RFC9397] section 6.2.1.

unneeded-manifest-list
: The unneeded-manifest-list parameter enumerates the SUIT manifests whose components are currently installed in the TEE, but which are no longer needed by any other application. The TAM can use this information in determining whether a SUIT manifest can be unlinked. Each unneeded SUIT manifest is identified by its SUIT Manifest Component ID (note that this is the Component ID for the manifest itself, which is different from the Component ID of a component installed by the manifest, see [I-D.ietf-suit-trust-domains] for more discussion). A TEEP Agent can get this information from the UnrequestTA conceptual API defined in [RFC9397] section 6.2.1.

ext-list
: The ext-list parameter lists the supported extensions. This document does not define any extensions. This parameter MUST be present if the QueryResponse is sent in response to a QueryRequest with the extensions bit set.

The requested-tc-info message has the following fields:

component-id
: A SUIT Component Identifier.

tc-manifest-sequence-number
: The minimum suit-manifest-sequence-number value from a SUIT manifest for the Trusted Component. If not present, indicates that any sequence number will do.

have-binary

If present with a value of true, indicates that the TEEP Agent already has the Trusted Component binary and only needs an Update message with a SUIT manifest that authorizes installing it. If have-binary is true, the tc-manifest-sequence-number field MUST be present.

4.3.1. Evidence and Attestation Results

Section 7 of [RFC9397] lists information that may appear in Evidence depending on the circumstance. However, the Evidence is opaque to the TEEP protocol and there are no formal requirements on the contents of Evidence.

TAMs however consume Attestation Results and do need enough information therein to make decisions on how to remediate a TEE that is out of compliance, or update a TEE that is requesting an authorized change. To do so, the information in Section 7 of [RFC9397] is often required depending on the policy.

Attestation Results SHOULD use Entity Attestation Tokens (EATs). Use of any other format, such as a widely implemented format for a specific processor vendor, is permitted but increases the complexity of the TAM by requiring it to understand the format for each such format rather than only the common EAT format so is not recommended.

When an EAT is used, the following claims can be used to meet those requirements, whether these claims appear in Attestation Results, or in Evidence for the Verifier to use when generating Attestation Results of some form:

Requirement	Claim	Reference
Freshness proof	nonce	Section 4.1 of [I-D.ietf-rats-eat]
Device unique identifier	ueid	Section 4.2.1 of [I-D.ietf-rats-eat]
Vendor of the device	oemid	Section 4.2.3 of [I-D.ietf-rats-eat]
Class of the device	hwmodel	Section 4.2.4 of [I-D.ietf-rats-eat]
TEE hardware type	hwversion	Section 4.2.5 of [I-D.ietf-rats-eat]
TEE hardware version	hwversion	Section 4.2.5 of [I-D.ietf-rats-eat]
TEE firmware type	manifests	Section 4.2.15 of [I-D.ietf-rats-eat]
TEE firmware version	manifests	Section 4.2.15 of [I-D.ietf-rats-eat]

Table 1

The "manifests" claim (see Section 4.2.15 of [I-D.ietf-rats-eat]) should include information about the TEEP Agent as well as any of its dependencies such as firmware.

4.4. Update Message

The Update message is used by the TAM to install and/or delete one or more Trusted Components via the TEEP Agent. It can also be used to pass a successful Attestation Report back to the TEEP Agent when the TAM is configured as an intermediary between the TEEP Agent and a Verifier, as shown in the figure below, where the Attestation Result passed back to the Attester can be used as a so-called "passport" (see section 5.1 of [RFC9334]) that can be presented to other Relying Parties.

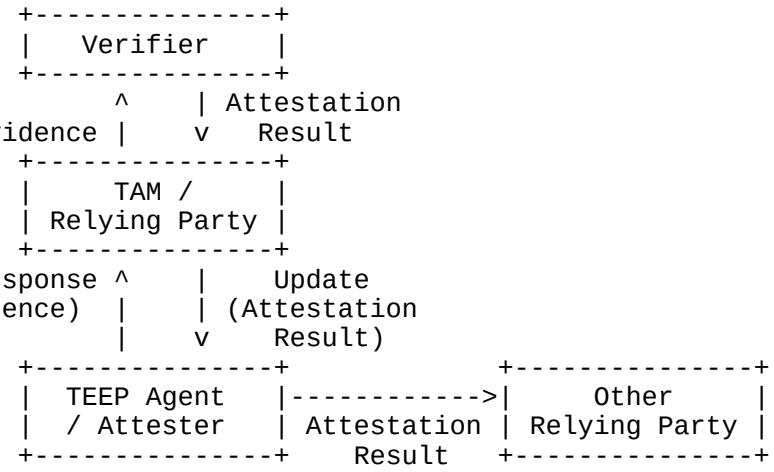


Figure 1: Example use of TEEP and attestation

Like other TEEP messages, the Update message is signed, and the relevant CDDL snippet is shown below. The complete CDDL structure is shown in Appendix C.

```
update = [
  type: TEEP-TYPE-update,
  options: {
    ? token => bstr .size (8..64),
    ? unneeded-manifest-list => [ + SUIT_Component_Identifier ],
    ? manifest-list => [ + bstr .cbor SUIT_Envelope ],
    ? attestation-payload-format => text,
    ? attestation-payload => bstr,
    ? err-code => (0..23),
    ? err-msg => text .size (1..128),
    * $$update-extensions,
    * $$teep-option-extensions
  }
]
```

The Update message has the following fields:

type
: The value of (3) corresponds to an Update message sent from the TAM to the TEEP Agent. In case of successful processing, a Success message is returned by the TEEP Agent. In case of an error, an Error message is returned. Note that the Update message is used for initial Trusted Component installation as well as for updates and deletes.

token
The value in the token field is used to match responses to requests.

unneeded-manifest-list
The unneeded-manifest-list parameter enumerates the SUIT manifests to be unlinked. Each unneeded SUIT manifest is identified by its SUIT Manifest Component ID. The SUIT manifest processor MAY execute uninstall section in the manifest. See Section 7 of [I-D.ietf-suit-trust-domains] for more information about the suit-uninstall Command Sequence.

manifest-list
The manifest-list field is used to convey one or multiple SUIT manifests to install. A manifest is a bundle of metadata about a Trusted Component, such as where to find the code, the devices to which it applies, and cryptographic information protecting the manifest. The manifest may also convey personalization data. Trusted Component binaries and personalization data can be signed and encrypted by the same Trusted Component Signer. Other combinations are, however, possible as well. For example, it is also possible for the TAM to sign and encrypt the personalization data and to let the Trusted Component Developer sign and/or encrypt the Trusted Component binary.

attestation-payload-format
The attestation-payload-format parameter indicates the IANA Media Type of the attestation-payload parameter, where media type parameters are permitted after the media type. The absence of this parameter indicates that the format is "application/eat+cwt; eat_profile=urn:ietf:rfc:rfcXXXX" (see [I-D.ietf-rats-eat-media-type] for further discussion). (RFC-editor: upon RFC publication, replace XXXX above with the RFC number of this document.) It MUST be present if the attestation-payload parameter is present and the format is not an EAT in CWT format with the profile defined below in Section 5.

attestation-payload
The attestation-payload parameter contains an Attestation Result. This parameter If the attestation-payload-format parameter is absent, the attestation payload contained in this parameter MUST be an Entity Attestation Token following the encoding defined in [I-D.ietf-rats-eat]. See Section 4.3.1 for further discussion.

err-code
The err-code parameter contains one of the error codes listed in the Section 4.6, which describes the reasons for the error when performing QueryResponse in the TAM.

err-msg
: The err-msg parameter is human-readable diagnostic text that MUST be encoded using UTF-8 [RFC3629] in Net-Unicode format [RFC5198] with a maximum of 128 bytes.

Note that an Update message carrying one or more SUIT manifests will inherently involve multiple signatures, one by the TAM in the TEEP message and one from a Trusted Component Signer inside each manifest. This is intentional as they are for different purposes.

The TAM is what authorizes apps to be installed, updated, and deleted on a given TEE and so the TEEP signature is checked by the TEEP Agent at protocol message processing time. (This same TEEP security wrapper is also used on messages like QueryRequest so that Agents only send potentially sensitive data such as Evidence to trusted TAMs.)

The Trusted Component signer on the other hand is what authorizes the Trusted Component to actually run, so the manifest signature could be checked at install time or load (or run) time or both, and this checking is done by the TEE independent of whether TEEP is used or some other update mechanism. See section 5 of [RFC9397] for further discussion.

The Update Message has a SUIT_Envelope containing SUIT manifests. Following are some example scenarios using SUIT manifests in the Update Message.

4.4.1. Scenario 1: Having one SUIT Manifest pointing to a URI of a Trusted Component Binary

In this scenario, a SUIT Manifest has a URI pointing to a Trusted Component Binary.

A Trusted Component Developer creates a new Trusted Component Binary and hosts it at a Trusted Component Developer’s URI. Then the Trusted Component Developer generates an associated SUIT manifest with the filename "tc-uuid" that contains the URI. The filename "tc-uuid" is used in Scenario 3 later.

The TAM receives the latest SUIT manifest from the Trusted Component Developer, and the URI it contains will not be changeable by the TAM since the SUIT manifest is signed by the Trusted Component Developer.

Pros:

* The Trusted Component Developer can ensure that the intact Trusted Component Binary is downloaded by devices

* The TAM does not have to send large Update messages containing the Trusted Component Binary

Cons:

* The Trusted Component Developer must host the Trusted Component Binary server

* The device must fetch the Trusted Component Binary in another connection after receiving an Update message

* A device’s IP address and therefore location may be revealed to the Trusted Component Binary server

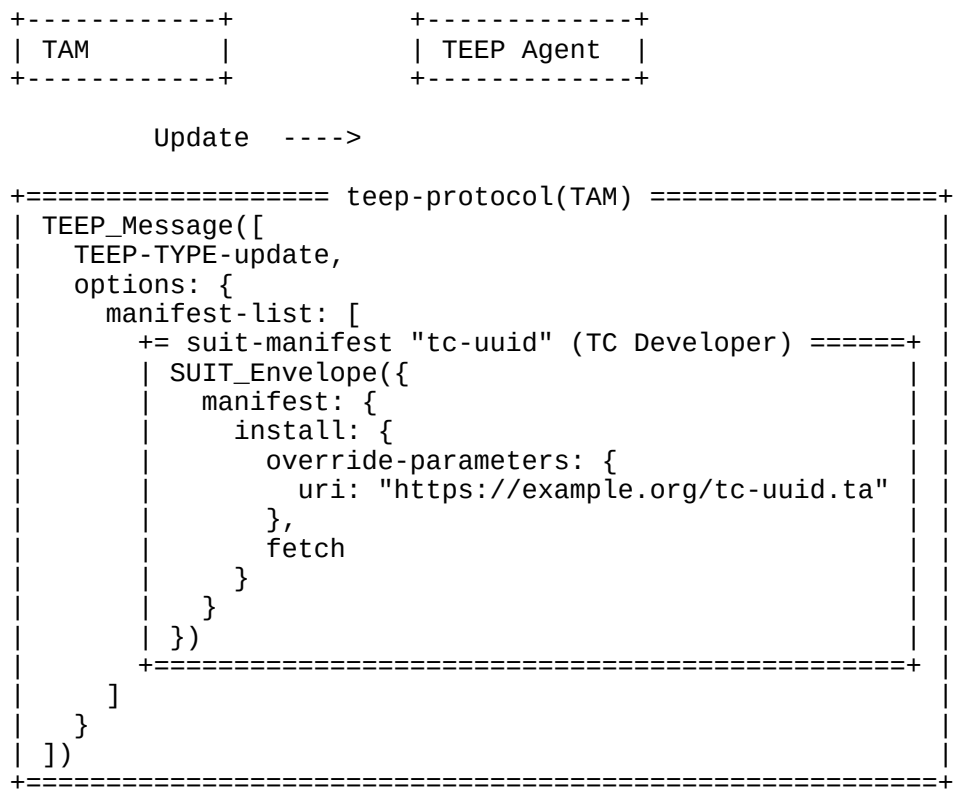


Figure 2: URI of the Trusted Component Binary

For the full SUIT Manifest example binary, see Appendix "Example 1: SUIT Manifest pointing to URI of the Trusted Component Binary".

4.4.2. Scenario 2: Having a SUIT Manifest include the Trusted Component Binary

In this scenario, the SUIT manifest contains the entire Trusted Component Binary as an integrated payload (see [I-D.ietf-suit-manifest] Section 7.5).

A Trusted Component Developer delegates the task of delivering the Trusted Component Binary to the TAM inside the SUIT manifest. The Trusted Component Developer creates a SUIT manifest and embeds the Trusted Component Binary, which is referenced in the suit-integrated-payload element containing the fragment-only reference "#tc", in the envelope. The Trusted Component Developer transmits the entire bundle to the TAM.

The TAM serves the SUIT manifest containing the Trusted Component Binary to the device in an Update message.

Pros:

* The device can obtain the Trusted Component Binary and the SUIT manifest in one Update message.

* The Trusted Component Developer does not have to host a server to deliver the Trusted Component Binary to devices.

Cons:

* The TAM must host the Trusted Component Binary rather than delegating storage to the Trusted Component Developer.

* The TAM must deliver Trusted Component Binaries in Update messages, which increases the size of the Update message.

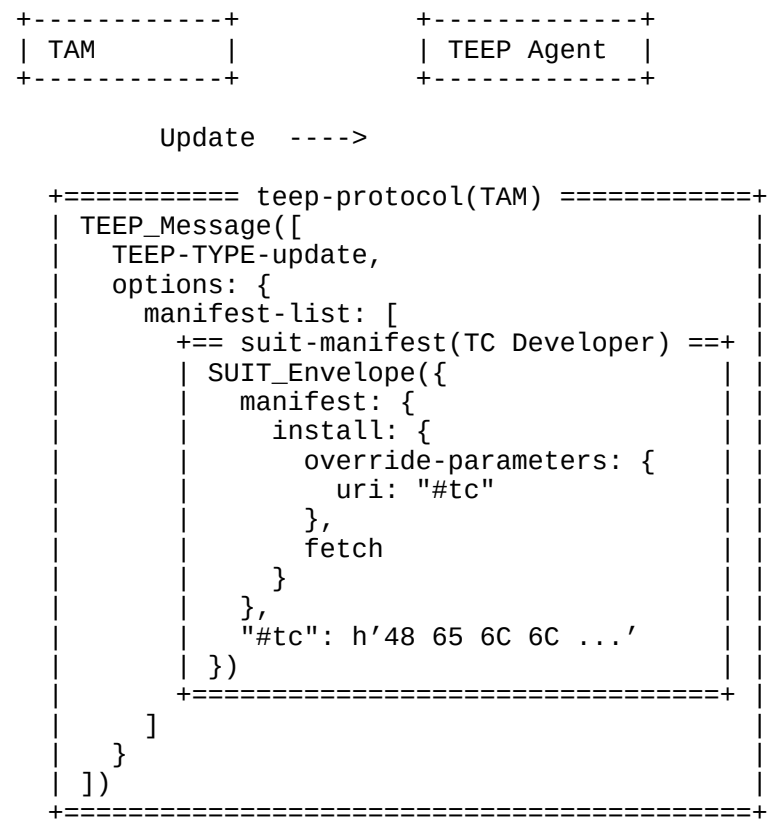


Figure 3: Integrated Payload with Trusted Component Binary

For the full SUIT Manifest example binary, see Appendix "Example 2: SUIT Manifest including the Trusted Component Binary".

4.4.3. Scenario 3: Supplying Personalization Data for the Trusted Component Binary

In this scenario, Personalization Data is associated with the Trusted Component Binary "tc-uuid" from Scenario 1.

The Trusted Component Developer places encrypted Personalization Data in the SUIT manifest, and it will be delivered by the TAM. The SUIT manifest processor decrypts it and then store it into file named "config.json", and then install the dependency component.

The TAM delivers the SUIT manifest of the Personalization Data which depends on the Trusted Component Binary from Scenario 1.

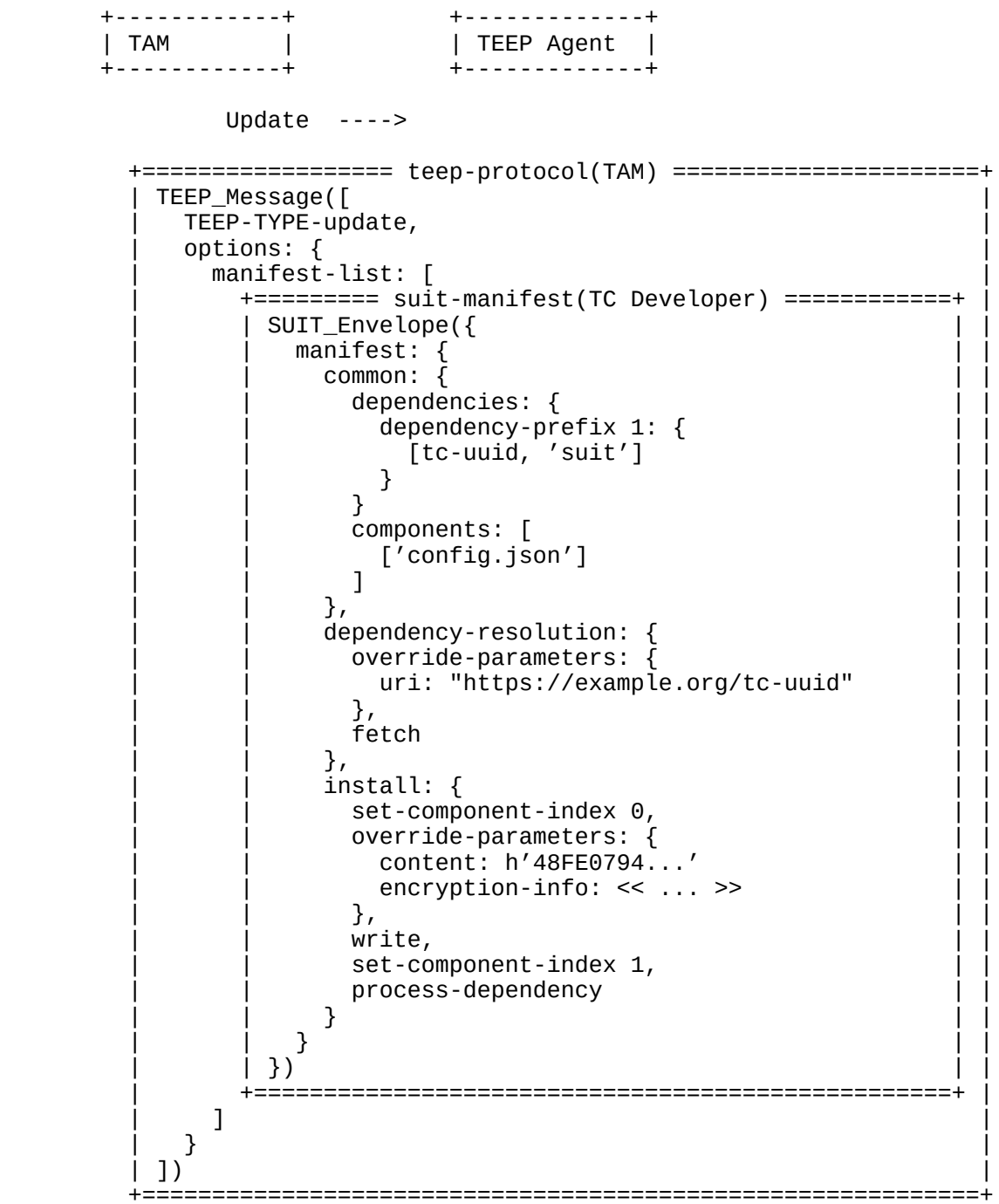

+------------+ +-------------+
| TAM | | TEEP Agent |
+------------+ +-------------+
Update ---->
+================== teep-protocol(TAM) ======================+
| TEEP_Message([|
| TEEP-TYPE-update, |
| options: { |
| manifest-list: [|
| +========== suit-manifest(TC Developer) ============+ |
	SUIT_Envelope({	
	manifest: {	
	common: {	
	dependencies: {	
	dependency-prefix 1: {	
	[tc-uuid, 'suit']	
	}	
	}	
	components: [	
	['config.json']	
	]	
	},	
	dependency-resolution: {	
	override-parameters: {	
	uri: "https://example.org/tc-uuid"	
	},	
	fetch	
	},	
	install: {	
	set-component-index 0,	
	override-parameters: {	
	content: h'48FE0794...'	
	encryption-info: << ... >>	
	},	
	write,	
	set-component-index 1,	
	process-dependency	
	}	
	}	
	})	
+===+
]
}
])
+==+

Figure 4: Encrypted Personalization Data

For the full SUIT Manifest example binary, see Appendix "Example 3: Supplying Personalization Data for Trusted Component Binary".

4.5. Success Message

The Success message is used by the TEEP Agent to return a success in response to an Update message.

Like other TEEP messages, the Success message is signed, and the relevant CDDL snippet is shown below. The complete CDDL structure is shown in Appendix C.

```
teep-success = [
  type: TEEP-TYPE-teep-success,
  options: {
    ? token => bstr .size (8..64),
    ? msg => text .size (1..128),
    ? suit-reports => [ + SUIT_Report ],
    * $$teep-success-extensions,
    * $$teep-option-extensions
  }
]
```

The Success message has the following fields:

type
: The value of (5) corresponds to corresponds to a Success message sent from the TEEP Agent to the TAM.

token
: The value in the token parameter is used to match responses to requests. It MUST match the value of the token parameter in the Update message the Success is in response to, if one was present. If none was present, the token MUST be absent in the Success message.

msg
: The msg parameter contains optional diagnostics information encoded in UTF-8 [RFC3629] using Net-Unicode form [RFC5198] with max 128 bytes returned by the TEEP Agent.

suit-reports

If present, the suit-reports parameter contains a set of SUIT Reports as defined in Section 4 of [I-D.ietf-suit-report]. If a token parameter was present in the Update message the Success message is in response to, the suit-report-nonce field MUST be present in the SUIT Report with a value matching the token parameter in the Update message.

4.6. Error Message

The Error message is used by the TEEP Agent to return an error in response to a message from the TAM.

Like other TEEP messages, the Error message is signed, and the relevant CDDL snippet is shown below. The complete CDDL structure is shown in Appendix C.

```
teep-error = [
  type: TEEP-TYPE-teep-error,
  options: {
     ? token => bstr .size (8..64),
     ? err-msg => text .size (1..128),
     ? supported-teep-cipher-suites => [ + $teep-cipher-suite ],
     ? supported-freshness-mechanisms => [ + $freshness-mechanism ],
     ? supported-suit-cose-profiles => [ + $suit-cose-profile ],
     ? challenge => bstr .size (8..512),
     ? versions => [ + version ],
     ? suit-reports => [ + SUIT_Report ],
     * $$teep-error-extensions,
     * $$teep-option-extensions
  },
  err-code: (0..23)
]

; The err-code parameter, uint (0..23)
ERR_PERMANENT_ERROR = 1
ERR_UNSUPPORTED_EXTENSION = 2
ERR_UNSUPPORTED_FRESHNESS_MECHANISMS = 3
ERR_UNSUPPORTED_MSG_VERSION = 4
ERR_UNSUPPORTED_CIPHER_SUITES = 5
ERR_BAD_CERTIFICATE = 6
ERR_ATTESTATION_REQUIRED = 7
ERR_UNSUPPORTED_SUIT_REPORT = 8
ERR_CERTIFICATE_EXPIRED = 9
ERR_TEMPORARY_ERROR = 10
ERR_MANIFEST_PROCESSING_FAILED = 17
```

The Error message has the following fields:

type
The value of (6) corresponds to an Error message sent from the TEEP Agent to the TAM.

token
The value in the token parameter is used to match responses to requests. It MUST match the value of the token parameter in the message the Success is in response to, if one was present. If none was present, the token MUST be absent in the Error message.

err-msg
The err-msg parameter is human-readable diagnostic text that MUST be encoded using UTF-8 [RFC3629] using Net-Unicode form [RFC5198] with max 128 bytes.

supported-teep-cipher-suites
The supported-teep-cipher-suites parameter lists the TEEP cipher suite(s) supported by the TEEP Agent. Details about the cipher suite encoding can be found in Section 8.1. This otherwise optional parameter MUST be returned if err-code is ERR_UNSUPPORTED_CIPHER_SUITES.

supported-freshness-mechanisms
The supported-freshness-mechanisms parameter lists the freshness mechanism(s) supported by the TEEP Agent. Details about the encoding can be found in Section 9. This otherwise optional parameter MUST be returned if err-code is ERR_UNSUPPORTED_FRESHNESS_MECHANISMS.

supported-suit-cose-profiles
The supported-suit-cose-profiles parameter lists the SUIT profiles supported by the TEEP Agent. Details about the cipher suite encoding can be found in Section 8.2. This otherwise optional parameter MUST be returned if err-code is ERR_UNSUPPORTED_SUIT_REPORT.

challenge
The challenge field is an optional parameter used for ensuring the freshness of attestation Evidence included with a QueryRequest message. When a challenge is provided in the Error message and Evidence in the form of an EAT is returned with a QueryRequest message then the challenge contained in the Error message MUST be used to generate the EAT, by copying the challenge value into the eat_nonce claim, as described in the EAT profile Section 5, if the nonce-based freshness mechanism is used. For more details see Section 9.

If any format other than EAT is used, it is up to that format to define the use of the challenge field.

versions
: The versions parameter enumerates the TEEP protocol version(s) supported by the TEEP Agent. This otherwise optional parameter MUST be returned if err-code is ERR_UNSUPPORTED_MSG_VERSION.

suit-reports
: If present, the suit-reports parameter contains a set of SUIT Reports as defined in Section 4 of [I-D.ietf-suit-report]. If a token parameter was present in the Update message the Error message is in response to, the suit-report-nonce field MUST be present in the SUIT Report with a value matching the token parameter in the Update message.

err-code
: The err-code parameter contains one of the error codes listed below). Only selected values are applicable to each message.

This specification defines the following initial error messages:

ERR_PERMANENT_ERROR (1)
: The received TEEP message contained incorrect fields or fields that are inconsistent with other fields. For diagnosis purposes it is RECOMMMENDED to identify the failure reason in the error message field. A TEEP implementation receiving this error might refuse to communicate further with the problematic TEEP message sender, by silently dropping any TEEP messages received, for some period of time until it has reason to believe it is worth trying again, but it should take care not to give up on communication. In contrast, ERR_TEMPORARY_ERROR is an indication that a more aggressive retry is warranted.

ERR_UNSUPPORTED_EXTENSION (2)
: The TEEP implementation does not support an extension included in the TEEP message it received. For diagnosis purposes it is RECOMMMENDED to identify the unsupported extension in the error message field. A TAM implementation receiving this error might retry sending the last message it sent to the sender of this error, without using any TEEP extensions.

ERR_UNSUPPORTED_FRESHNESS_MECHANISMS (3)
: The TEEP Agent does not support any freshness algorithm mechanisms in the request message. A TAM receiving this error might retry the request using a different set of supported freshness mechanisms in the request message.

ERR_UNSUPPORTED_MSG_VERSION (4)
The TEEP implementation does not support the TEEP protocol version indicated in the received message. A TAM receiving this error might retry the request using a different TEEP protocol version.

ERR_UNSUPPORTED_CIPHER_SUITES (5)
The TEEP Agent does not support any cipher suites indicated in the request message. A TAM receiving this error might retry the request using a different set of supported cipher suites in the request message.

ERR_BAD_CERTIFICATE (6)
Processing of a certificate failed. For diagnosis purposes it is RECOMMMENDED to include information about the failing certificate in the error message field. For example, the certificate was of an unsupported type, or the certificate was revoked by its signer. A TEEP implementation receiving this error might attempt to use an alternate certificate.

ERR_ATTESTATION_REQUIRED (7)
Indicates that the TEEP implementation sending this error requires attestation of the TEEP imlementation receiving this error.

ERR_UNSUPPORTED_SUIT_REPORT (8)
Indicates that the TEEP Agent does not support the suit-cose-profile of the SUIT Reports which was sent by the TAM. The TEEP Agent must report the error code ERR_UNSUPPORTED_SUIT_REPORT supplying the supported-suit-cose-profiles.

ERR_CERTIFICATE_EXPIRED (9)
A certificate has expired or is not currently valid. A TEEP implementation receiving this error might attempt to renew its certificate before using it again.

ERR_TEMPORARY_ERROR (10)
A miscellaneous temporary error, such as a memory allocation failure, occurred while processing the TEEP message. A TEEP implementation receiving this error might retry the last message it sent to the sender of this error at some later point, which is up to the implementation.

ERR_MANIFEST_PROCESSING_FAILED (17)
The TEEP Agent encountered one or more manifest processing failures. If the suit-reports parameter is present, it contains the failure details. A TAM receiving this error might still attempt to install or update other components that do not depend on the failed manifest.

New error codes should be added sparingly, not for every implementation error. That is the intent of the err-msg field, which can be used to provide details meaningful to humans. New error codes should only be added if the TAM is expected to do something behaviorally different upon receipt of the error message, rather than just logging the event. Hence, each error code is responsible for saying what the behavioral difference is expected to be.

5. EAT Profile

The TEEP protocol operates between a TEEP Agent and a TAM. While the TEEP protocol does not require use of EAT, use of EAT is encouraged and Section 4.3 explicitly defines a way to carry an Entity Attestation Token in a QueryResponse.

As discussed in Section 4.3.1, the content of Evidence is opaque to the TEEP architecture, but the content of Attestation Results is not, where Attestation Results flow between a Verifier and a TAM (as the Relying Party). Although Attestation Results required by a TAM are separable from the TEEP protocol per se, this section is included as part of the requirements for building a compliant TAM that uses EATs for Attestation Results.

Section 7 of [I-D.ietf-rats-eat] defines the requirement for Entity Attestation Token profiles. This section defines an EAT profile for use with TEEP.

* profile-label: The profile-label for this specification is the URI

<urn:ietf:rfc:rfcXXXX>. (RFC-editor: upon RFC publication, replace XXXX with the RFC number of this document.)

* Use of JSON, CBOR, or both: CBOR only.

* CBOR Map and Array Encoding: Only definite length arrays and maps.

* CBOR String Encoding: Only definite-length strings are allowed.

* CBOR Preferred Serialization: Encoders must use preferred serialization, and decoders need not accept non-preferred serialization.

* CBOR Tags: CBOR Tags are not used.

* COSE/JOSE Protection: See Section 8.2.

* COSE/JOSE Algorithms: See Section 8.2.

* Detached EAT Bundle Support: DEB use is permitted.

* Key Identification: COSE Key ID (kid) is used, where the key ID is the hash of a public key (where the public key may be used as a raw public key, or in a certificate) as specified in [I-D.ietf-cose-key-thumbprint]. See Section 7.1.1.1 and Section 7.2.1.1 for discussion on the choice of hash algorithm.

* Endorsement Identification: Optional, but semantics are the same as in Verification Key Identification.

* Freshness: See Section 9 for details. When the eat_nonce claim is used, the value is a single bstr.

* Claims Requirements:

  - The following claims are required: ueid, oemid, hwmodel, hwversion, manifests, and cnf. See Section 4.3.1 for discussion. Other claims are optional.

  - See Section 9 for discussion affecting whether the eat_nonce claim is used.

  - The sw-name claim for a Trusted Component holds the URI of the SUIT manifest for that component.

  - The manifests claim uses a SUIT manifest, where the manifest body contains a SUIT_Reference as defined in Section 4 of [I-D.ietf-suit-report], and the content type is as defined in [I-D.ietf-suit-report].

A TAM implementation might simply accept a TEEP Agent as trustworthy based on a successful Attestation Result, and if not then attempt to update the TEEP Agent and all of its dependencies. This logic is simple but it might result in updating some components that do not need to be updated.

An alternate TAM implementation might use any Additional Claims to determine whether the TEEP Agent or any of its dependencies are trustworthy, and only update the specific components that are out of date.

5.1. Relationship to AR4SI

[I-D.ietf-rats-ar4si] defines an EAT profile for arbitrary Relying Parties to use with Attestation Results. However the TAM as a Relying Party needs specific claims that are not required in the AR4SI profile, and so needs its own more specific profile.

In some deployments, a TAM can be used as an intermediary between Verifier and a TEEP Agent acting as an Attester in the Passport model or acting as a Relying Party in the Background Check Model of [RFC9334]. This is depicted in the example in Figure 1. In such a case, both profiles need to be obtained from the Verifier: one for use by the TAM itself, and the other to pass on to the TEEP Agent.

When the TAM and Verifier are combined into the same implementation, obtaining both profiles can be straightforward, but when they are on different machines, the situation is more complex, especially if Nonces are used to ensure freshness of Evidence. There are thus several such cases:

1. The protocol between the TAM and the Verifier (which is outside the scope of TEEP itself) allows requesting multiple Attestation Results from the same Evidence. In this case, the TAM can request both EAT profiles be returned.

2. The protocol between the TAM and the Verifier only allows requesting one Attestation Result format, but the Evidence freshness mechanism does not use Nonces. In this case, the TAM can send the same Evidence in two separate requests, each requesting a different EAT profile for the Attestation Results.

3. The protocol between the TAM and the Verifier only allows requesting one Attestation Result format, and the Evidence freshness mechanism uses Nonces. In this case, it is simpler to not have the TAM be an intermediary, since the Verifier will require a separate Nonce for each Attestation Result, but have the Attester or Relying Party contact the Verifier directly to get Attestation Results in the AR4SI profile.

6. Mapping of TEEP Message Parameters to CBOR Labels

In COSE, arrays and maps use strings, negative integers, and unsigned integers as their keys. Integers are used for compactness of encoding. Since the word "key" is mainly used in its other meaning, as a cryptographic key, this specification uses the term "label" for this usage as a map key.

This specification uses the following mapping:

Name	Label
supported-teep-cipher-suites	1
challenge	2
versions	3
supported-suit-cose-profiles	4
selected-version	6
attestation-payload	7
tc-list	8
ext-list	9
manifest-list	10
msg	11
err-msg	12
attestation-payload-format	13
requested-tc-list	14
unneeded-manifest-list	15
component-id	16
tc-manifest-sequence-number	17
have-binary	18
suit-reports	19
token	20
supported-freshness-mechanisms	21
err-code	23

Table 2

```
; labels of mapkey for teep message parameters, uint (0..23)
supported-teep-cipher-suites = 1
challenge = 2
versions = 3
supported-suit-cose-profiles = 4
selected-version = 6
attestation-payload = 7
tc-list = 8
ext-list = 9
manifest-list = 10
msg = 11
err-msg = 12
attestation-payload-format = 13
requested-tc-list = 14
unneeded-manifest-list = 15
component-id = 16
tc-manifest-sequence-number = 17
have-binary = 18
suit-reports = 19
token = 20
supported-freshness-mechanisms = 21
err-code = 23
```

7. Behavior Specification

Behavior is specified in terms of the conceptual APIs defined in section 6.2.1 of [RFC9397].

7.1. TAM Behavior

When the ProcessConnect API is invoked, the TAM sends a QueryRequest message.

When the ProcessTeepMessage API is invoked, the TAM first does validation as specified in Section 4.1.2, and drops the message if it is not valid. It may also do additional implementation specific actions such as logging the results or attempting to update the TEEP Agent to a version that does not send invalid messages. Otherwise, it proceeds as follows.

If the message includes a token, it can be used to match the response to a request previously sent by the TAM. The TAM MUST expire the token value after receiving the first response from the device that has a valid signature and ignore any subsequent messages that have the same token value. The token value MUST NOT be used for other purposes, such as a TAM to identify the devices and/or a device to identify TAMs or Trusted Components.

7.1.1. Handling a QueryResponse Message

If a QueryResponse message is received, the TAM verifies the presence of any parameters required based on the data-items-requested in the QueryRequest, and also validates that the nonce in any SUIT Report matches the token sent in the QueryRequest message if a token was present. If these requirements are not met, the TAM drops the message and sends an Update message containing an appropriate err-code and err-msg. It may also do additional implementation specific actions such as logging the results. If the requirements are met, processing continues as follows.

If a QueryResponse message is received that contains an attestation-payload, the TAM checks whether it contains Evidence or an Attestation Result by inspecting the attestation-payload-format parameter. The media type defined in Section 5 indicates an Attestation Result, though future extensions might also indicate other Attestation Result formats in the future. Any other unrecognized value indicates Evidence. If it contains an Attestation Result, processing continues as in Section 7.1.1.1.

If the QueryResponse is instead determined to contain Evidence, the TAM passes the Evidence (via some mechanism out of scope of this document) to an attestation Verifier (see [RFC9334]) to determine whether the Agent is in a trustworthy state. Once the TAM receives an Attestation Result from the Verifier, processing continues as in Section 7.1.1.1.

7.1.1.1. Handling an Attestation Result

The Attestation Result must first be validated as follows:

1. Verify that the Attestation Result was signed by a Verifier that the TAM trusts.

2. Verify that the Attestation Result contains a "cnf" claim (as defined in Section 3.1 of [RFC8747]) where the key ID is the hash of the TEEP Agent public key used to verify the signature on the TEEP message, and the hash is computed using the Digest Algorithm specified by one of the SUIT profiles supported by the TAM (SHA-256 for the ones mandated in this document).

   See Sections 3.4 and 6 of [RFC8747] for more discussion.

Based on the results of attestation (if any), any SUIT Reports, and the lists of installed, requested, and unneeded Trusted Components reported in the QueryResponse, the TAM determines, in any implementation specific manner, which Trusted Components need to be installed, updated, or deleted, if any. There are in typically three cases:

1. Attestation failed. This indicates that the rest of the information in the QueryResponse cannot necessarily be trusted, as the TEEP Agent may not be healthy (or at least up to date). In this case, the TAM might attempt to use TEEP to update any Trusted Components (e.g., firmware, the TEEP Agent itself, etc.) needed to get the TEEP Agent back into an up-to-date state that would allow attestation to succeed. If the TAM does not have permission to update such components (this can happen if different TAMs manage different components in the device), the TAM instead responds with an Update message containing an appropriate err-msg, and err-code set to ERR_ATTESTATION_REQUIRED.

2. Attestation succeeded (so the QueryResponse information can be accepted as valid), but the set of Trusted Components needs to be updated based on TAM policy changes or requests from the TEEP Agent.

3. Attestation succeeded, and no changes are needed.

If any Trusted Components need to be installed, updated, or deleted, the TAM sends an Update message containing SUIT Manifests with command sequences to do the relevant installs, updates, or deletes. It is important to note that the TEEP Agent’s Update Procedure requires resolving and installing any dependencies indicated in the manifest, which may take some time, and the resulting Success or Error message is generated only after completing the Update Procedure. Hence, depending on the freshness mechanism in use, the TAM may need to store data (e.g., a nonce) for some time. For example, if a mobile device needs an unmetered connection to download a dependency, it may take hours or longer before the device has sufficient access. A different freshness mechanism, such as timestamps, might be more appropriate in such cases.

If no Trusted Components need to be installed, updated, or deleted, but the QueryResponse included Evidence, the TAM MAY (e.g., based on attestation-payload-format parameters received from the TEEP Agent in the QueryResponse) still send an Update message with no SUIT Manifests, to pass the Attestation Result back to the TEEP Agent.

7.1.2. Handling a Success or Error Message

If a Success or Error message is received containing one or more SUIT Reports, the TAM also validates that the nonce in any SUIT Report matches the token sent in the Update message, and drops the message if it does not match. Otherwise, the TAM handles the update in any implementation specific way, such as updating any locally cached information about the state of the TEEP Agent, or logging the results.

If an Error message is received with the error code ERR_ATTESTATION_REQUIRED, it indicates that the TEEP Agent is requesting attestation of the TAM. In this case, the TAM MUST send another QueryRequest with an attestation-payload and optionally a suit-report to the TEEP Agent.

If any other Error message is received, the TAM can handle it in any implementation specific way, but Section 4.6 provides recommendations for such handling.

7.2. TEEP Agent Behavior

When the RequestTA API is invoked, the TEEP Agent first checks whether the requested TA is already installed. If it is already installed, the TEEP Agent passes no data back to the caller. Otherwise, if the TEEP Agent chooses to initiate the process of requesting the indicated TA, it determines (in any implementation specific way) the TAM URI based on any TAM URI provided by the RequestTA caller and any local configuration, and passes back the TAM URI to connect to. It MAY also pass back a QueryResponse message if all of the following conditions are true:

* The last QueryRequest message received from that TAM contained no token or challenge,

* The ProcessError API was not invoked for that TAM since the last QueryResponse message was received from it, and

* The public key or certificate of the TAM is cached and not expired.

When the RequestPolicyCheck API is invoked, the TEEP Agent decides whether to initiate communication with any trusted TAMs (e.g., it might choose to do so for a given TAM unless it detects that it has already communicated with that TAM recently). If so, it passes back a TAM URI to connect to. If the TEEP Agent has multiple TAMs it needs to connect with, it just passes back one, with the expectation that RequestPolicyCheck API will be invoked to retrieve each one

successively until there are no more and it can pass back no data at that time. Thus, once a TAM URI is returned, the TEEP Agent can remember that it has already initiated communication with that TAM.

When the ProcessError API is invoked, the TEEP Agent can handle it in any implementation specific way, such as logging the error or using the information in future choices of TAM URI.

When the ProcessTeepMessage API is invoked, the Agent first does validation as specified in Section 4.1.2, and if it is not valid then the Agent responds with an Error message. Otherwise, processing continues as follows based on the type of message.

7.2.1. Handling a QueryRequest Message

When a QueryRequest message is received, it is processed as follows.

If the TEEP Agent requires attesting the TAM and the QueryRequest message did not contain an attestation-payload, the TEEP Agent MUST send an Error Message with the error code ERR_ATTESTATION_REQUIRED supplying the supported-freshness-mechanisms and challenge if needed. Otherwise, processing continues as follows.

If the TEEP Agent requires attesting the TAM and the QueryRequest message did contain an attestation-payload, the TEEP Agent checks whether it contains Evidence or an Attestation Result by inspecting the attestation-payload-format parameter. The media type defined in Section 5 indicates an Attestation Result, though future extensions might also indicate other Attestation Result formats in the future. Any other unrecognized value indicates Evidence. If it contains an Attestation Result, processing continues as in Section 7.2.1.1.

If the QueryRequest is instead determined to contain Evidence, the TEEP Agent passes the Evidence (via some mechanism out of scope of this document) to an attestation Verifier (see [RFC9334]) to determine whether the TAM is in a trustworthy state. Once the TEEP Agent receives an Attestation Result from the Verifier, processing continues as in Section 7.2.1.1.

The TEEP Agent MAY also use (in any implementation specific way) any SUIT Reports in the QueryRequest in determining whether it trusts the TAM. If a SUIT Report uses a suit-cose-profile that the TEEP Agent does not support, then the TEEP Agent MUST send an Error Message with the error code ERR_UNSUPPORTED_SUIT_REPORT supplying the supported-suit-cose-profiles. Otherwise, processing continues as follows.

Once the Attestation Result is handled, or if the TEEP Agent does not require attesting the TAM, the Agent responds with a QueryResponse message if all fields were understood, or an Error message if any error was encountered.

7.2.1.1. Handling an Attestation Result

The Attestation Result must first be validated as follows:

1. Verify that the Attestation Result was signed by a Verifier that the TEEP Agent trusts.

2. Verify that the Attestation Result contains a "cnf" claim (as defined in Section 3.1 of [RFC8747]) where the key ID is the hash of the TAM public key used to verify the signature on the TEEP message, and the hash is computed using the Digest Algorithm specified by one of the SUIT profiles supported by the TEEP Agent (SHA-256 for the ones mandated in this document).

   See Sections 3.4 and 6 of [RFC8747] for more discussion.

7.2.2. Handling an Update Message

When an Update message is received, the Agent attempts to unlink any SUIT manifests listed in the unneeded-manifest-list field of the message, and responds with an Error message if any error was encountered. If the unneeded-manifest-list was empty, or no error was encountered processing it, the Agent attempts to update the Trusted Components specified in the SUIT manifests by following the Update Procedure specified in [I-D.ietf-suit-manifest], and responds with a Success message if all SUIT manifests were successfully installed, or an Error message if any error was encountered. It is important to note that the Update Procedure requires resolving and installing any dependencies indicated in the manifest, which may take some time, and the Success or Error message is generated only after completing the Update Procedure.

8. Cipher Suites

TEEP requires algorithms for various purposes:

* Algorithms for signing TEEP messages exchanged between the TEEP Agent and the TAM.

* Algorithms for signing EAT-based Evidence sent by the Attester via the TEEP Agent and the TAM to the Verifier.

* Algorithms for encrypting EAT-based Evidence sent by the TEEP Agent to the TAM. (The TAM will decrypt the encrypted Evidence and will forward it to the Verifier.)

* Algorithms for signing and optionally encrypting SUIT reports sent by the TEEP Agent to the TAM.

* Algorithms for signing and optionally encrypting SUIT manifests sent by the Trusted Component Signer to the TEEP Agent.

Further details are provided for the protection of TEEP messages, SUIT Reports, and EATs.

8.1. TEEP Messages

The TEEP protocol uses COSE for protection of TEEP messages in both directions. To negotiate cryptographic mechanisms and algorithms, the TEEP protocol defines the following cipher suite structure, which is used to specify an ordered set of operations (e.g., sign) done as part of composing a TEEP message. Although this specification only specifies the use of signing and relies on payload encryption to protect sensitive information, future extensions might specify support for encryption and/or MAC operations if needed.

```
; teep-cipher-suites
$teep-cipher-suite /= teep-cipher-suite-sign1-eddsa
$teep-cipher-suite /= teep-cipher-suite-sign1-es256

;The following two cipher suites have only a single operation each.
;Other cipher suites may be defined to have multiple operations.
;It is MANDATORY for TAM to support them, and OPTIONAL
;to support any additional ones that use COSE_Sign_Tagged, or other
;signing, encryption, or MAC algorithms.

teep-operation-sign1-eddsa = [ cose-sign1, cose-alg-eddsa ]
teep-operation-sign1-es256 = [ cose-sign1, cose-alg-es256 ]

teep-cipher-suite-sign1-eddsa = [ teep-operation-sign1-eddsa ]
teep-cipher-suite-sign1-es256 = [ teep-operation-sign1-es256 ]

;MANDATORY for TAM and TEEP Agent to support the following COSE
;operations, and OPTIONAL to support additional ones such as
;COSE_Sign_Tagged, COSE_Encrypt0_Tagged, etc.

cose-sign1 = 18      ; CoAP Content-Format value

;MANDATORY for TAM to support the following, and OPTIONAL to implement
;any additional algorithms from the IANA COSE Algorithms registry.

cose-alg-es256 = -7  ; ECDSA w/ SHA-256
cose-alg-eddsa = -8  ; EdDSA
```

Each operation in a given cipher suite has two elements:

* a COSE-type defined in Section 2 of [RFC9052] that identifies the type of operation, and

* a specific cryptographic algorithm as defined in the COSE Algorithms registry [COSE.Algorithm] to be used to perform that operation.

A TAM MUST support both of the cipher suites defined above. A TEEP Agent MUST support at least one of the two but can choose which one. For example, a TEEP Agent might choose a given cipher suite if it has hardware support for it. A TAM or TEEP Agent MAY also support any other algorithms in the COSE Algorithms registry in addition to the mandatory ones listed above. It MAY also support use with COSE_Sign or other COSE types in additional cipher suites.

Any cipher suites without confidentiality protection can only be added if the associated specification includes a discussion of security considerations and applicability, since manifests may carry

sensitive information. For example, Section 6 of [RFC9397] permits implementations that terminate transport security inside the TEE and if the transport security provides confidentiality then additional encryption might not be needed in the manifest for some use cases. For most use cases, however, manifest confidentiality will be needed to protect sensitive fields from the TAM as discussed in Section 9.8 of [RFC9397].

The cipher suites defined above do not do encryption at the TEEP layer, but permit encryption of the SUIT payload using a mechanism such as [I-D.ietf-suit-firmware-encryption]. See Section 10 and Section 8.2 for more discussion of specific payloads.

For the initial QueryRequest message, unless the TAM has more specific knowledge about the TEEP Agent (e.g., if the QueryRequest is sent in response to some underlying transport message that contains a hint), the message does not use COSE_Sign1 with one of the above cipher suites, but instead uses COSE_Sign with multiple signatures, one for each algorithm used in any of the cipher suites listed in the supported-teep-cipher-suites parameter of the QueryRequest, so that a TEEP Agent supporting any one of them can verify a signature. If the TAM does have specific knowledge about which cipher suite the TEEP Agent supports, it MAY instead use that cipher suite with the QueryRequest.

For an Error message with code ERR_UNSUPPORTED_CIPHER_SUITES, the TEEP Agent MUST protect it with any of the cipher suites mandatory for the TAM.

For all other TEEP messages between the TAM and TEEP Agent, the selected TEEP cipher suite MUST be used in both directions.

8.2. EATs and SUIT Reports

TEEP uses COSE for confidentiality of EATs and SUIT Reports sent by a TEEP Agent. The TEEP Agent obtains a signed EAT and then SHOULD encrypt it using the TAM as the recipient. A SUIT Report is created by a SUIT processor, which is part of the TEEP Agent itself. The TEEP Agent is therefore in control of signing the SUIT Report and SHOULD encrypt it. Again, the TAM is the recipient of the encrypted content. For content-key distribution Ephemeral-Static Diffie-Hellman (ES-DH) is used in this specification. See Section 8.5.5 and Appendix B of [RFC9052] for more details. (If [I-D.ietf-suit-firmware-encryption] is used, it is also the same as discussed in Section 6.2 of that document.)

ES-DH is a scheme that provides public key encryption given a recipient's public key. Hence, the TEEP Agent needs to be in possession of the public key of the TAM. See Section 5 of [RFC9397] for more discussion of TAM keys used by the TEEP Agent. There are multiple variants of this scheme; this document uses the variant specified in Section 8.5.5 of [RFC9052].

The following two layer structure is used:

* Layer 0: Has a content encrypted with the Content Encryption Key (CEK), a symmetric key. For encrypting SUIT Reports and EATs the content MUST NOT be detached.

* Layer 1: Uses the AES Key Wrap algorithm to encrypt the randomly generated CEK with the Key Encryption Key (KEK) derived with ES-DH, whereby the resulting symmetric key is fed into the HKDF-based key derivation function.

As a result, the two layers combine ES-DH with AES-KW and HKDF.

This document re-uses the CDDL defined in Section 6.2.3 of [I-D.ietf-suit-firmware-encryption] and the context information structure defined in Section 6.2.4 of [I-D.ietf-suit-firmware-encryption] although with an important modification. The COSE_KDF_Context.SuppPubInfo.other value MUST be set to "SUIT Report Encryption" when a SUIT Report is encrypted and MUST be set to "EAT Encryption" when an EAT is encrypted. The COSE_KDF_Context.SuppPubInfo.other field captures the protocol in which the ES-DH content key distribution algorithm is used.

This specification defines cipher suites for confidentiality protection of EATs and SUIT Reports. The TAM MUST support each cipher suite defined below, based on definitions in [I-D.ietf-suit-mti]. A TEEP Agent MUST support at least one of the cipher suites below but can choose which one. For example, a TEEP Agent might choose a given cipher suite if it has hardware support for it. A TAM or TEEP Agent MAY also support other algorithms in the COSE Algorithms registry. It MAY also support use with COSE_Encrypt or other COSE types in additional cipher suites.

```
; suit-cose-profile
$suit-cose-profile /= suit-sha256-es256-ecdh-a128ctr
$suit-cose-profile /= suit-sha256-eddsa-ecdh-a128ctr
$suit-cose-profile /= suit-sha256-es256-ecdh-a128gcm
$suit-cose-profile /= suit-sha256-eddsa-ecdh-chacha-poly
```

9. Attestation Freshness Mechanisms

A freshness mechanism determines how a TAM can tell whether an attestation payload provided in a QueryResponse is fresh. There are multiple ways this can be done as discussed in Section 10 of [RFC9334].

Each freshness mechanism is identified with an integer value, which corresponds to an IANA registered freshness mechanism (see the IANA Considerations section of [I-D.ietf-rats-reference-interaction-models]). This document uses the following freshness mechanisms which may be added to in the future by TEEP extensions:

```
; freshness-mechanisms
FRESHNESS_NONCE = 0
FRESHNESS_TIMESTAMP = 1

$freshness-mechanism /= FRESHNESS_NONCE
$freshness-mechanism /= FRESHNESS_TIMESTAMP
```

An implementation MUST support the Nonce mechanism and MAY support additional mechanisms.

In the Nonce mechanism, the attestation payload MUST include a nonce provided in the QueryRequest challenge if the Background Check model is used, or in the QueryRequest token if the Passport model is used. The timestamp mechanism uses a timestamp determined via mechanisms outside the TEEP protocol, and the challenge is only needed in the QueryRequest message if a challenge is needed in generating the attestation payload for reasons other than freshness.

If a TAM supports multiple freshness mechanisms that require different challenge formats, the QueryRequest message can currently only send one such challenge. This situation is expected to be rare, but should it occur, the TAM can choose to prioritize one of them and exclude the other from the supported-freshness-mechanisms in the QueryRequest, and resend the QueryRequest with the other mechanism if an ERR_UNSUPPORTED_FRESHNESS_MECHANISMS Error is received that indicates the TEEP Agent supports the other mechanism.

10. Security Considerations

This section summarizes the security considerations discussed in this specification:

Cryptographic Algorithms

TEEP protocol messages exchanged between the TAM and the TEEP Agent are protected using COSE. This specification relies on the cryptographic algorithms provided by COSE. Public key based authentication is used by the TEEP Agent to authenticate the TAM and vice versa.

Attestation
: A TAM relies on signed Attestation Results provided by a Verifier, either obtained directly using a mechanism outside the TEEP protocol (by using some mechanism to pass Evidence obtained in the attestation payload of a QueryResponse, and getting back the Attestation Results), or indirectly via the TEEP Agent forwarding the Attestation Results in the attestation payload of a QueryResponse. See the security considerations of the specific mechanism in use (e.g., EAT) for more discussion.

  An impersonation attack, where one TEEP Agent attempts to use the attestation payload of another TEEP Agent, can be prevented using a proof-of-possession approach. The "cnf" claim is mandatory in the EAT profile for EAT for this purpose. See Section 6 of [RFC8747] and Section 7.1.1.1 and Section 7.2.1.1 of this document for more discussion.

Trusted Component Binaries
: Each Trusted Component binary is signed by a Trusted Component Signer. It is the responsibility of the TAM to relay only verified Trusted Components from authorized Trusted Component Signers. Delivery of a Trusted Component to the TEEP Agent is then the responsibility of the TAM, using the security mechanisms provided by the TEEP protocol. To protect the Trusted Component binary, the SUIT manifest format is used and it offers a variety of security features, including digital signatures and content encryption, if a SUIT mechanism such as [I-D.ietf-suit-firmware-encryption] is used.

Personalization Data
: A Trusted Component Signer or TAM can supply personalization data along with a Trusted Component. This data is also protected by a SUIT manifest. Personalization data is signed and encrypted by a Trusted Component Signer, if a SUIT mechanism such as [I-D.ietf-suit-firmware-encryption] is used.

TEEP Broker
: As discussed in section 6 of [RFC9397], the TEEP protocol typically relies on a TEEP Broker to relay messages between the TAM and the TEEP Agent. When the TEEP Broker is compromised it can drop messages, delay the delivery of messages, and replay messages but it cannot modify those messages. (A replay would be,

however, detected by the TEEP Agent.) A compromised TEEP Broker could reorder messages in an attempt to install an old version of a Trusted Component. Information in the manifest ensures that TEEP Agents are protected against such downgrade attacks based on features offered by the manifest itself.

Replay Protection
: The TEEP protocol supports replay protection as follows. The transport protocol under the TEEP protocol might provide replay protection, but may be terminated in the TEEP Broker which is not trusted by the TEEP Agent and so the TEEP protocol does replay protection itself. If attestation of the TAM is used, the attestation freshness mechanism provides replay protection for attested QueryRequest messages. If non-attested QueryRequest messages are replayed, the TEEP Agent will generate QueryResponse or Error messages, but the REE can already conduct Denial of Service attacks against the TEE and/or the TAM even without the TEEP protocol. QueryResponse messages have replay protection via attestation freshness mechanism, or the token field in the message if attestation is not used. Update messages have replay protection via the suit-manifest-sequence-number (see Section 8.4.2 of [I-D.ietf-suit-manifest]). Error and Success messages have replay protection via SUIT Reports and/or the token field in the message, where a TAM can detect which message it is in response to.

Trusted Component Signer Compromise
: A TAM is responsible for vetting a Trusted Component and before distributing them to TEEP Agents.
It is RECOMMENDED to provide a way to update the trust anchor store used by the TEE, for example using a firmware update mechanism such as [I-D.ietf-rats-concise-ta-stores]. Thus, if a Trusted Component Signer is later compromised, the TAM can update the trust anchor store used by the TEE, for example using a firmware update mechanism.

CA Compromise
: The CA issuing certificates to a TEE or a Trusted Component Signer might get compromised. It is RECOMMENDED to provide a way to update the trust anchor store used by the TEE, for example by using a firmware update mechanism, Concise TA Stores [I-D.ietf-rats-concise-ta-stores], Trust Anchor Management Protocol (TAMP) [RFC5934] or a similar mechanism. If the CA issuing certificates to devices gets compromised then these devices will be rejected by a TAM, if revocation is available to the TAM.

TAM Certificate Expiry

The integrity and the accuracy of the clock within the TEE determines the ability to determine an expired TAM certificate, if certificates are used.

Compromised Time Source
: As discussed above, certificate validity checks rely on comparing validity dates to the current time, which relies on having a trusted source of time, such as [RFC8915]. A compromised time source could thus be used to subvert such validity checks.

11. Privacy Considerations

Depending on the properties of the attestation mechanism, it is possible to uniquely identify a device based on information in the attestation payload or in the certificate used to sign the attestation payload. This uniqueness may raise privacy concerns. To lower the privacy implications the TEEP Agent MUST present its attestation payload only to an authenticated and authorized TAM and when using an EAT, it SHOULD use encryption as discussed in [I-D.ietf-rats-eat], since confidentiality is not provided by the TEEP protocol itself and the transport protocol under the TEEP protocol might be implemented outside of any TEE. If any mechanism other than EAT is used, it is up to that mechanism to specify how privacy is provided.

Since SUIT Reports can also contain sensitive information, a TEEP Agent SHOULD also encrypt SUIT Reports as discussed in Section 8.2.

In addition, in the usage scenario discussed in Section 4.4.1, a device reveals its IP address to the Trusted Component Binary server. This can reveal to that server at least a clue as to its location, which might be sensitive information in some cases.

EATs and SUIT Reports from a TAM can also be present in a QueryRequest. Typically, the ability to uniquely identify a TAM is less of a concern than it is for TEEP Agents, but where confidentiality is a concern for the TAM, such EATs and SUIT Reports SHOULD be encrypted just like ones from TEEP Agents.

12. IANA Considerations

12.1. Media Type Registration

IANA is requested to assign a media type for application/teep+cbor.

Type name: application

Subtype name: teep+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Same as encoding considerations of application/cbor.

Security considerations: See Security Considerations Section of this document.

Interoperability considerations: Same as interoperability considerations of application/cbor as specified in [RFC8949].

Published specification: This document.

Applications that use this media type: TEEP protocol implementations

Fragment identifier considerations: N/A

Additional information: Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person to contact for further information: teep@ietf.org

Intended usage: COMMON

Restrictions on usage: none

Author: See the "Authors' Addresses" section of this document

Change controller: IETF

13. References

13.1. Normative References

[COSE.Algorithm]
IANA, "COSE Algorithms", n.d., <https://www.iana.org/assignments/cose/cose.xhtml#algorithms>.

[I-D.ietf-cose-key-thumbprint]
Isobe, K., Tschofenig, H., and O. Steele, "CBOR Object Signing and Encryption (COSE) Key Thumbprint", Work in Progress, Internet-Draft, draft-ietf-cose-key-thumbprint-06, 6 September 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-key-thumbprint-06>.

[I-D.ietf-rats-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>.

[I-D.ietf-suit-manifest]
Moran, B., Tschofenig, H., Birkholz, H., Zandberg, K., and O. R淡nningstad, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", Work in Progress, Internet-Draft, draft-ietf-suit-manifest-33, 24 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-manifest-33>.

[I-D.ietf-suit-mti]
Moran, B., R淡nningstad, O., and A. Tsukamoto, "Mandatory-to-Implement Algorithms for Authors and Recipients of Software Update for the Internet of Things manifests", Work in Progress, Internet-Draft, draft-ietf-suit-mti-09, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-mti-09>.

[I-D.ietf-suit-report]
Moran, B. and H. Birkholz, "Secure Reporting of Update Status", Work in Progress, Internet-Draft, draft-ietf-suit-report-11, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-report-11>.

[I-D.ietf-suit-trust-domains]
Moran, B. and K. Takayama, "SUIT Manifest Extensions for Multiple Trust Domains", Work in Progress, Internet-Draft, draft-ietf-suit-trust-domains-10, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-trust-domains-10>.

[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>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/rfc/rfc3629>.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <https://www.rfc-editor.org/rfc/rfc5198>.

[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>.

[RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <https://www.rfc-editor.org/rfc/rfc8747>.

[RFC8949] 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>.

[RFC9052] 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>.

13.2. Informative References

[I-D.ietf-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-08, 6 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-ar4si-08>.

[I-D.ietf-rats-concise-ta-stores] Wallace, C., Housley, R., Fossati, T., and Y. Deshpande, "Concise TA Stores (CoTS)", Work in Progress, Internet-Draft, draft-ietf-rats-concise-ta-stores-02, 5 December 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-concise-ta-stores-02>.

[I-D.ietf-rats-eat-media-type]
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>.

[I-D.ietf-rats-reference-interaction-models]
Birkholz, H., Eckel, M., Pan, W., and E. Voit, "Reference Interaction Models for Remote Attestation Procedures", Work in Progress, Internet-Draft, draft-ietf-rats-reference-interaction-models-13, 26 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-reference-interaction-models-13>.

[I-D.ietf-suit-firmware-encryption]
Tschofenig, H., Housley, R., Moran, B., Brown, D., and K. Takayama, "Encrypted Payloads in SUIT Manifests", Work in Progress, Internet-Draft, draft-ietf-suit-firmware-encryption-23, 29 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-firmware-encryption-23>.

[RFC5934] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Management Protocol (TAMP)", RFC 5934, DOI 10.17487/RFC5934, August 2010, <https://www.rfc-editor.org/rfc/rfc5934>.

[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>.

[RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020, <https://www.rfc-editor.org/rfc/rfc8915>.

[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>.

[RFC9397] Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", RFC 9397, DOI 10.17487/RFC9397, July 2023, <https://www.rfc-editor.org/rfc/rfc9397>.

A. Contributors

We would like to thank Brian Witten (Symantec), Tyler Kim (Solacia), Nick Cook (Arm), and Minho Yoo (IoTrust) for their contributions to the Open Trust Protocol (OTrP), which influenced the design of this specification.

B. Acknowledgements

We would like to thank Eve Schooler for the suggestion of the protocol name.

We would like to thank Kohei Isobe (TRASIO/SECOM), Ken Takayama (SECOM), Kuniyasu Suzaki (TRASIO/AIST), Tsukasa Oi (TRASIO), and Yuichi Takita (SECOM) for their valuable implementation feedback.

We would also like to thank Carsten Bormann and Henk Birkholz for their help with the CDDL.

C. Complete CDDL

Valid TEEP messages adhere to the following CDDL data definitions, except that SUIT_Envelope and SUIT_Component_Identifier are specified in [I-D.ietf-suit-manifest].

This section is informative and merely summarizes the normative CDDL snippets in the body of this document.

```
; DO NOT EDIT this cddl file manually.
; This cddl file is Auto-generated file from md file.
; Edit the md file and run make for generating this cddl file.
; Please do not forget to commit and push this cddl file to git repo
; every time you have revised the md file.

teep-message = $teep-message-type .within teep-message-framework

teep-message-framework = [
  type: $teep-type / $teep-type-extension,
  options: { * teep-option },
  * any; further elements, e.g., for data-item-requested
]

teep-option = (uint => any)

; messages defined below:
$teep-message-type /= query-request
$teep-message-type /= query-response
$teep-message-type /= update
```

```
$teep-message-type /= teep-success
$teep-message-type /= teep-error

; message type numbers, in one byte which could take a number from 0 to 23
$teep-type = (0..23)
TEEP-TYPE-query-request = 1
TEEP-TYPE-query-response = 2
TEEP-TYPE-update = 3
TEEP-TYPE-teep-success = 5
TEEP-TYPE-teep-error = 6

query-request = [
  type: TEEP-TYPE-query-request,
  options: {
    ? token => bstr .size (8..64),
    ? supported-freshness-mechanisms => [ + $freshness-mechanism ],
    ? challenge => bstr .size (8..512),
    ? versions => [ + version ],
    ? attestation-payload-format => text,
    ? attestation-payload => bstr,
    ? suit-reports => [ + bstr ],
    * $$query-request-extensions,
    * $$teep-option-extensions
  },
  supported-teep-cipher-suites: [ + $teep-cipher-suite ],
  supported-suit-cose-profiles: [ + $suit-cose-profile ],
  data-item-requested: uint .bits data-item-requested
]

version = uint .size 4
ext-info = uint .size 4

; data items as bitmaps
data-item-requested = &(
  attestation: 0,
  trusted-components: 1,
  extensions: 2,
  suit-reports: 3,
)

; teep-cipher-suites
$teep-cipher-suite /= teep-cipher-suite-sign1-eddsa
$teep-cipher-suite /= teep-cipher-suite-sign1-es256

;The following two cipher suites have only a single operation each.
;Other cipher suites may be defined to have multiple operations.
;It is MANDATORY for TAM to support them, and OPTIONAL
;to support any additional ones that use COSE_Sign_Tagged, or other
```

```
   ;signing, encryption, or MAC algorithms.

   teep-operation-sign1-eddsa = [ cose-sign1, cose-alg-eddsa ]
   teep-operation-sign1-es256 = [ cose-sign1, cose-alg-es256 ]

   teep-cipher-suite-sign1-eddsa = [ teep-operation-sign1-eddsa ]
   teep-cipher-suite-sign1-es256 = [ teep-operation-sign1-es256 ]

   ;MANDATORY for TAM and TEEP Agent to support the following COSE
   ;operations, and OPTIONAL to support additional ones such as
   ;COSE_Sign_Tagged, COSE_Encrypt0_Tagged, etc.

   cose-sign1 = 18      ; CoAP Content-Format value

   ;MANDATORY for TAM to support the following, and OPTIONAL to implement
   ;any additional algorithms from the IANA COSE Algorithms registry.

   cose-alg-es256 = -7  ; ECDSA w/ SHA-256
   cose-alg-eddsa = -8  ; EdDSA

   ; suit-cose-profile
   $suit-cose-profile /= suit-sha256-es256-ecdh-a128ctr
   $suit-cose-profile /= suit-sha256-eddsa-ecdh-a128ctr
   $suit-cose-profile /= suit-sha256-es256-ecdh-a128gcm
   $suit-cose-profile /= suit-sha256-eddsa-ecdh-chacha-poly

   ; freshness-mechanisms
   FRESHNESS_NONCE = 0
   FRESHNESS_TIMESTAMP = 1

   $freshness-mechanism /= FRESHNESS_NONCE
   $freshness-mechanism /= FRESHNESS_TIMESTAMP

   query-response = [
     type: TEEP-TYPE-query-response,
     options: {
       ? token => bstr .size (8..64),
       ? selected-version => version,
       ? attestation-payload-format => text,
       ? attestation-payload => bstr,
       ? suit-reports => [ + bstr ],
       ? tc-list => [ + system-property-claims ],
       ? requested-tc-list => [ + requested-tc-info ],
       ? unneeded-manifest-list => [ + SUIT_Component_Identifier ],
       ? ext-list => [ + ext-info ],
       * $$query-response-extensions,
       * $$teep-option-extensions
     }
```

```
   ]

   requested-tc-info = {
     component-id => SUIT_Component_Identifier,
     ? tc-manifest-sequence-number => uint .size 8,
     ? have-binary => bool
   }

   update = [
     type: TEEP-TYPE-update,
     options: {
       ? token => bstr .size (8..64),
       ? unneeded-manifest-list => [ + SUIT_Component_Identifier ],
       ? manifest-list => [ + bstr .cbor SUIT_Envelope ],
       ? attestation-payload-format => text,
       ? attestation-payload => bstr,
       ? err-code => (0..23),
       ? err-msg => text .size (1..128),
       * $$update-extensions,
       * $$teep-option-extensions
     }
   ]

   teep-success = [
     type: TEEP-TYPE-teep-success,
     options: {
       ? token => bstr .size (8..64),
       ? msg => text .size (1..128),
       ? suit-reports => [ + SUIT_Report ],
       * $$teep-success-extensions,
       * $$teep-option-extensions
     }
   ]

   teep-error = [
     type: TEEP-TYPE-teep-error,
     options: {
        ? token => bstr .size (8..64),
        ? err-msg => text .size (1..128),
        ? supported-teep-cipher-suites => [ + $teep-cipher-suite ],
        ? supported-freshness-mechanisms => [ + $freshness-mechanism ],
        ? supported-suit-cose-profiles => [ + $suit-cose-profile ],
        ? challenge => bstr .size (8..512),
        ? versions => [ + version ],
        ? suit-reports => [ + SUIT_Report ],
        * $$teep-error-extensions,
        * $$teep-option-extensions
     },
```

```
     err-code: (0..23)
   ]

   ; The err-code parameter, uint (0..23)
   ERR_PERMANENT_ERROR = 1
   ERR_UNSUPPORTED_EXTENSION = 2
   ERR_UNSUPPORTED_FRESHNESS_MECHANISMS = 3
   ERR_UNSUPPORTED_MSG_VERSION = 4
   ERR_UNSUPPORTED_CIPHER_SUITES = 5
   ERR_BAD_CERTIFICATE = 6
   ERR_ATTESTATION_REQUIRED = 7
   ERR_UNSUPPORTED_SUIT_REPORT = 8
   ERR_CERTIFICATE_EXPIRED = 9
   ERR_TEMPORARY_ERROR = 10
   ERR_MANIFEST_PROCESSING_FAILED = 17

   ; labels of mapkey for teep message parameters, uint (0..23)
   supported-teep-cipher-suites = 1
   challenge = 2
   versions = 3
   supported-suit-cose-profiles = 4
   selected-version = 6
   attestation-payload = 7
   tc-list = 8
   ext-list = 9
   manifest-list = 10
   msg = 11
   err-msg = 12
   attestation-payload-format = 13
   requested-tc-list = 14
   unneeded-manifest-list = 15
   component-id = 16
   tc-manifest-sequence-number = 17
   have-binary = 18
   suit-reports = 19
   token = 20
   supported-freshness-mechanisms = 21
   err-code = 23
```

D. Examples of Diagnostic Notation and Binary Representation

This section includes some examples with the following assumptions:

* The device will have two TCs with the following SUIT Component Identifiers:

  - [0x000102030405060708090a0b0c0d0e0f]

```
      -  [ 0x100102030405060708090a0b0c0d0e0f ]
```

* SUIT manifest-list is set empty only for example purposes (see Appendix E for actual manifest examples)

D.1. QueryRequest Message

D.1.1. CBOR Diagnostic Notation

```
   / query-request = /
   [
     / type: / 1 / TEEP-TYPE-query-request /,
     / options: /
     {
       / token / 20 : h'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF',
       / versions / 3 : [ 0 ]  / 0 is current TEEP Protocol /
     },
     / supported-teep-cipher-suites: / [
       [ [ 18, -7 ] ] / Sign1 using ES256 /,
       [ [ 18, -8 ] ] / Sign1 using EdDSA /
     ],
     / supported-suit-cose-profiles: / [
       [-16, -7, -29, -65534] / suit-sha256-es256-ecdh-a128ctr /,
       [-16, -8, -29, -65534] / suit-sha256-eddsa-ecdh-a128ctr /,
       [-16, -7, -29, 1]      / suit-sha256-es256-ecdh-a128gcm /,
       [-16, -8, -29, 24]     / suit-sha256-eddsa-ecdh-chacha-poly /
     ],
     / data-item-requested: / 3 / attestation | trusted-components /
   ]
```

D.1.2. CBOR Binary Representation

```
  85                  # array(5)
     01               # unsigned(1) / TEEP-TYPE-query-request /
     A2               # map(2)
        14            # unsigned(20) / token: /
        50            # bytes(16)
           A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
        03            # unsigned(3) / versions: /
        81            # array(1) / [ 0 ] /
           00         # unsigned(0)
     82               # array(2) / supported-teep-cipher-suites /
        81            # array(1)
           82         # array(2)
              12      # unsigned(18) / cose-sign1 /
              26      # negative(6) / -7 = cose-alg-es256 /
        81            # array(1)
           82         # array(2)
              12      # unsigned(18) / cose-sign1 /
              27      # negative(7) / -8 = cose-alg-eddsa /
     84               # array(4) / supported-suit-cose-profiles /
        84            # array(4) / suit-sha256-es256-ecdh-a128ctr /,
           2f         # negative(15) / -16 = SHA-256 /
           26         # negative(6) / -7 = ES256 /
           38 1C      # negative(28) / -29 = ECDH-ES + A128KW /
           39 fffd    # negative(65533) / -65534 = A128CTR /
        84            # array(4) / suit-sha256-eddsa-ecdh-a128ctr /
           2f         # negative(15) / -16 = SHA-256 /
           27         # negative(7) / -8 = EdDSA /
           38 1C      # negative(28) / -29 = ECDH-ES + A128KW /
           39 fffd    # negative(65533) / -65534 = A128CTR /
        84            # array(4) / suit-sha256-es256-ecdh-a128gcm /
           2f         # negative(15) / -16 = SHA-256 /
           26         # negative(6) / -7 = ES256 /
           38 1C      # negative(28) / -29 = ECDH-ES + A128KW /
           01         # unsigned(1) / A128GCM /
        84            # array(4) / suit-sha256-eddsa-ecdh-chacha-poly /
           2f         # negative(15) / -16 = SHA-256 /
           27         # negative(7) / EdDSA /
           38 1C      # negative(28) / -29 = ECDH-ES + A128KW /
           18 18      # unsigned(24) / 24 = ChaCha20/Poly1305 /
     03               # unsigned(3) / attestation | trusted-components /
```

D.2. Entity Attestation Token

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

D.2.1. CBOR Diagnostic Notation

```
/ eat-claim-set = /
{
    / cnf /          8: {
                         / kid / 3 : h'ba7816bf8f01cfea414140de5dae2223'
                                     h'b00361a396177a9cb410ff61f20015ad'
                       },
    / eat_nonce /   10: h'948f8860d13a463e8e',
    / ueid /       256: h'0198f50a4ff6c05861c8860d13a638ea',
    / oemid /      258: h'894823', / IEEE OUI format OEM ID /
    / hwmodel /    259: h'549dcecc8b987c737b44e40f7c635ce8'
                        / Hash of chip model name /,
    / hwversion /  260: ["1.3.4", 1], / Multipartnumeric  /
    / manifests /  273: [
                          [ 60, / application/cbor, TO BE REPLACED /
                                / with the format value for a /
                                / SUIT_Reference once one is allocated /
                            {   / SUIT_Reference /
                              / suit-report-manifest-uri / 1: "https://example.com/manife
st.cbor",
                              / suit-report-manifest-digest / 0:[
                                / algorithm-id / -16 / "sha256" /,
                                / digest-bytes / h'a7fd6593eac32eb4be578278e6540c5c'
                                                 h'09cfd7d4d234973054833b2b93030609'
                                ]
                            }
                          ]
                        ]
}
```

D.3. QueryResponse Message

D.3.1. CBOR Diagnostic Notation

```
/ query-response = /
[
  / type: / 2 / TEEP-TYPE-query-response /,
  / options: /
  {
    / token / 20 : h'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF',
    / selected-version / 6 : 0,
    / attestation-payload / 7 : h'' / empty only for example purpose /,
    / tc-list / 8 : [
      {
        / system-component-id / 0 : [ h'0102030405060708090A0B0C0D0E0F' ],
        / suit-parameter-image-digest / 3: << [
          / suit-digest-algorithm-id / -16 / SHA256 /,
          / suit-digest-bytes / h'A7FD6593EAC32EB4BE578278E6540C5C09CFD7D4D234973054833B2
B93030609'
            / SHA256 digest of tc binary /
        ] >>
      }
    ]
  }
]
```

D.3.2. CBOR Binary Representation

```
82                  # array(2)
   02               # unsigned(2) / TEEP-TYPE-query-response /
   A4               # map(4)
      14            # unsigned(20) / token: /
      50            # bytes(16)
         A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
      06            # unsigned(6) / selected-version: /
      00            # unsigned(0)
      07            # unsigned(7) / attestation-payload: /
      40            # bytes(0)
                    # ""
      08            # unsigned(8) / tc-list: /
      81            # array(1)
         A2         # map(2)
            00      # unsigned(0) / system-component-id: /
            81      # array(1)
               4F   # bytes(15)
                  0102030405060708090A0B0C0D0E0F
            03      # unsigned(3) / suit-parameter-image-digest: /
            58 24   # bytes(36)
               822F5820A7FD6593EAC32EB4BE578278E6540C5C09CFD7D4D234973054833B2B93030609
```

D.4. Update Message

D.4.1. CBOR Diagnostic Notation

```
/ update = /
[
  / type: / 3 / TEEP-TYPE-update /,
  / options: /
  {
    / token / 20 : h'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF',
    / manifest-list / 10 : [
      <<
        / SUIT_Envelope / {
          / suit-authentication-wrapper / 2: << [
            << [
              / suit-digest-algorithm-id: / -16 / suit-cose-alg-sha256 /,
              / suit-digest-bytes: / h'DB601ADE73092B58532CA03FBB663DE49532435336F1558B49
BB622726A2FEDD'
            ] >>,
            << / COSE_Sign1_Tagged / 18( [
              / protected: / << {
                / algorithm-id / 1: -7 / ES256 /
              } >>,
              / unprotected: / {},
              / payload: / null,
              / signature: / h'5B2D535A2B6D5E3C585C1074F414DA9E10BD285C99A33916DADE3ED388
12504817AC48B62B8E984EC622785BD1C411888BE531B1B594507816B201F6F28579A4'
            ] ) >>
          ] >>,
          / suit-manifest / 3: << {
            / suit-manifest-version / 1: 1,
            / suit-manifest-sequence-number / 2: 3,
            / suit-common / 3: << {
              / suit-components / 2: [
                [
                  h'544545502D446576696365',           / "TEEP-Device" /
                  h'5365637572654653',                 / "SecureFS" /
                  h'8D82573A926D4754935332DC29997F74', / tc-uuid /
                  h'7461'                              / "ta" /
                ]
              ],
              / suit-common-sequence / 4: << [
                / suit-directive-override-parameters / 20, {
                  / suit-parameter-vendor-identifier / 1: h'C0DDD5F15243566087DB4F5B0AA26
C2F',
                  / suit-parameter-class-identifier / 2: h'DB42F7093D8C55BAA8C5265FC5820F
4E',
                  / suit-parameter-image-digest / 3: << [
                    / suit-digest-algorithm-id: / -16 / suit-cose-alg-sha256 /,
                    / suit-digest-bytes: / h'8CF71AC86AF31BE184EC7A05A411A8C3A14FD9B77A30
D046397481469468ECE8'
                  ] >>,
                  / suit-parameter-image-size / 14: 20
```

```
                },
                / suit-condition-vendor-identifier / 1, 15,
                / suit-condition-class-identifier / 2, 15
              ] >>
            } >>,
            / suit-install / 9: << [
              / suit-directive-override-parameters / 20, {
                / suit-parameter-uri / 21: "https://example.org/8d82573a-926d-4754-9353-3
2dc29997f74.ta"
              },
              / suit-directive-fetch / 21, 15,
              / suit-condition-image-match / 3, 15
            ] >>
          } >>
        }
      >>
    ] / array of bstr wrapped SUIT_Envelope /
  }
]
```

D.4.2. CBOR Binary Representation

```
82                  # array(2)
   03               # unsigned(3) / TEEP-TYPE-update /
   A2               # map(2)
      14            # unsigned(20) / token: /
      50            # bytes(16)
         A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
      0A            # unsigned(10) / manifest-list: /
      81            # array(1)
         59 014E    # bytes(336)
            A2025873825824822F5820DB601ADE73092B58532CA03FBB663DE495
            32435336F1558B49BB622726A2FEDD584AD28443A10126A0F658405B2D53
            5A2B6D5E3C585C1074F414DA9E10BD285C99A33916DADE3ED38812504817
            AC48B62B8E984EC622785BD1C411888BE531B1B594507816B201F6F28579
            A40358D4A401010203035884A20281844B544545502D4465766963654853
            65637572654653508D82573A926D4754935332DC29997F74427461045854
            8614A40150C0DDD5F15243566087DB4F5B0AA26C2F0250DB42F7093D8C55
            BAA8C5265FC5820F4E035824822F58208CF71AC86AF31BE184EC7A05A411
            A8C3A14FD9B77A30D046397481469468ECE80E14010F020F0958458614A1
            15783B68747470733A2F2F6578616D706C652E6F72672F38643832353733
            612D393236642D343735342D393335332D333264633239393937663742E
            7461150F030F
```

D.5. Success Message

D.5.1. CBOR Diagnostic Notation

```
/ teep-success = /
[
  / type: / 5 / TEEP-TYPE-teep-success /,
  / options: /
  {
    / token / 20 : h'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'
  }
]
```

D.5.2. CBOR Binary Representation

```
82                  # array(2)
   05               # unsigned(5) / TEEP-TYPE-teep-success /
   A1               # map(1)
      14            # unsigned(20) / token: /
      50            # bytes(16)
         A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
```

D.6. Error Message

D.6.1. CBOR Diagnostic Notation

```
/ teep-error = /
[
  / type: / 6 / TEEP-TYPE-teep-error /,
  / options: /
  {
    / token / 20 : h'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF',
    / err-msg / 12 : "disk-full"
  },
  / err-code: / 17 / ERR_MANIFEST_PROCESSING_FAILED /
]
```

D.6.2. CBOR binary Representation

```
83                  # array(3)
   06               # unsigned(6) / TEEP-TYPE-teep-error /
   A2               # map(2)
      14            # unsigned(20) / token: /
      50            # bytes(16)
         A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
      0C            # unsigned(12) / err-msg: /
      69            # text(9)
         6469736B2D66756C6C # "disk-full"
   11               # unsigned(17) / ERR_MANIFEST_PROCESSING_FAILED /
```

E. Examples of SUIT Manifests

This section shows some examples of SUIT manifests described in Section 4.4.

The examples are signed using the following ECDSA secp256r1 key with SHA256 as the digest function.

COSE_Sign1 Cryptographic Key:

```
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgApZYjZCUGLM50VBC
CjYStX+09jGmnyJPrpDLTz/hiXOhRANCAASEloEarguqq9JhVxie7NomvqqL8Rtv
P+bitWWchdvArTsfKktsCYExwKNtrNHXi9OB3N+wnAUtszmR23M4tKiW
-----END PRIVATE KEY-----
```

The corresponding public key can be used to verify these examples:

```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhJaBGq4LqqvSYVcYnuzaJr6qi/Eb
bz/m4rVlnIXbwK07HypLbAmBMcCjbazR14vTgdzfsJwFLbM5kdtzOLSolg==
-----END PUBLIC KEY-----
```

Example 1: SUIT Manifest pointing to URI of the Trusted Component Binary

CBOR Diagnostic Notation of SUIT Manifest

```
/ SUIT_Envelope / {
  / authentication-wrapper / 2: << [
    << [
      / digest-algorithm-id: / -16 / SHA256 /,
      / digest-bytes: / h'B39B52B0B747EA79588C190F567BFC2C8437BA8A73F7EA983182E79F0148D59
B'
    ] >>,
    << / COSE_Sign1_Tagged / 18([
      / protected: / << {
        / algorithm-id / 1: -7 / ES256 /
      } >>,
      / unprotected: / {},
      / payload: / null,
      / signature: / h'80E54AB485B320A61654666362928B15EAAABFE6957B1BCB65F16A367E4B19888B
FFDBD6F7EA2892FA36FA18A2FCB5DBFEC9832E09B91ED9CD348AB77E25FA74'
    ]) >>
  ] >>,
  / manifest / 3: << {
    / manifest-version / 1: 1,
    / manifest-sequence-number / 2: 3,
    / common / 3: << {
      / components / 2: [
        [
```

```
           'TEEP-Device',
           'SecureFS',
          h'8D82573A926D4754935332DC29997F74', / tc-uuid /
           'ta'
        ]
      ],
      / shared-sequence / 4: << [
        / directive-override-parameters / 20, {
          / parameter-vendor-identifier / 1: h'C0DDD5F15243566087DB4F5B0AA26C2F',
          / parameter-class-identifier / 2: h'DB42F7093D8C55BAA8C5265FC5820F4E',
          / parameter-image-digest / 3: << [
            / digest-algorithm-id: / -16 / SHA256 /,
            / digest-bytes: / h'8CF71AC86AF31BE184EC7A05A411A8C3A14FD9B77A30D046397481469
468ECE8'
          ] >>,
          / parameter-image-size / 14: 20
        },
        / condition-vendor-identifier / 1, 15,
        / condition-class-identifier / 2, 15
      ] >>
    } >>,
    / manifest-component-id / 5: [
       'TEEP-Device',
       'SecureFS',
      h'8D82573A926D4754935332DC29997F74',  / tc-uuid /
       'suit'
    ],
    / install / 20: << [
      / directive-override-parameters / 20, {
        / parameter-uri / 21: "https://example.org/8d82573a-926d-4754-9353-32dc29997f74.t
a"
      },
      / directive-fetch / 21, 15,
      / condition-image-match / 3, 15
    ] >>,
    / uninstall / 24: << [
      / directive-unlink / 33, 15
    ] >>
  } >>
}
```

CBOR Binary in Hex

```
   A2025873825824822F5820B39B52B0B747EA79588C190F567BFC2C8437BA
   8A73F7EA983182E79F0148D59B584AD28443A10126A0F6584080E54AB485
   B320A61654666362928B15EAAABFE6957B1BCB65F16A367E4B19888BFFDB
   D6F7EA2892FA36FA18A2FCB5DBFEC9832E09B91ED9CD348AB77E25FA7403
   590108A601010203035884A20281844B544545502D446576696365485365
   637572654653508D82573A926D4754935332DC29997F7442746104585486
   14A40150C0DDD5F15243566087DB4F5B0AA26C2F0250DB42F7093D8C55BA
   A8C5265FC5820F4E035824822F58208CF71AC86AF31BE184EC7A05A411A8
   C3A14FD9B77A30D046397481469468ECE80E14010F020F05844B54454550
   2D4465766963654853656375726546535 08D82573A926D4754935332DC29
   997F7444737569741458458614A115783B68747470733A2F2F6578616D70
   6C652E6F72672F38643832353733612D393236642D343735342D39333533
   2D333264633239393937663742E7461150F030F1818448218210F
```

Example 2: SUIT Manifest including the Trusted Component Binary

CBOR Diagnostic Notation of SUIT Manifest

```
/ SUIT_Envelope / {
  / authentication-wrapper / 2: << [
    << [
      / digest-algorithm-id: / -16 / SHA256 /,
      / digest-bytes: / h'CEDB0457952F7DD0A33FA4692F73BC833A6A6E2300B16F6605993F0192E3F21
9'
    ] >>,
    << / COSE_Sign1_Tagged / 18([
      / protected: / << {
        / algorithm-id / 1: -7 / ES256 /
      } >>,
      / unprotected: / {},
      / payload: / null,
      / signature: / h'71E3869E4E134A78C95D7ED81F5911FEA4F189EC33C0F6474C866569ED3DF7FB4E
0D8871367BA3C73612A26C9E3984A4E22CAA4BFBCE84DCAC0539AE87BE9D3D'
    ]) >>
  ] >>,
  / manifest / 3: << {
    / manifest-version / 1: 1,
    / manifest-sequence-number / 2: 3,
    / common / 3: << {
      / components / 2: [
        [
           'TEEP-Device',
           'SecureFS',
          h'8D82573A926D4754935332DC29997F74', / tc-uuid /
           'ta'
        ]
      ],
      / shared-sequence / 4: << [
        / directive-override-parameters / 20, {
          / parameter-vendor-identifier / 1: h'C0DDD5F15243566087DB4F5B0AA26C2F',
```

```
          / parameter-class-identifier / 2: h'DB42F7093D8C55BAA8C5265FC5820F4E',
          / parameter-image-digest / 3: << [
            / digest-algorithm-id: / -16 / SHA256 /,
            / digest-bytes: / h'8CF71AC86AF31BE184EC7A05A411A8C3A14FD9B77A30D046397481469
468ECE8'
          ] >>,
          / parameter-image-size / 14: 20
        },
        / condition-vendor-identifier / 1, 15,
        / condition-class-identifier / 2, 15
      ] >>
    } >>,
    / manifest-component-id / 5: [
       'TEEP-Device',
       'SecureFS',
      h'8D82573A926D4754935332DC29997F74',  / tc-uuid /
       'suit'
    ],
    / install / 20: << [
      / directive-override-parameters / 20, {
        / uri / 21: "#tc"
      },
      / directive-fetch / 21, 15,
      / condition-image-match / 3, 15
    ] >>,
    / uninstall / 24: << [
      / directive-unlink / 33, 15
    ] >>
  } >>,
  "#tc" : 'Hello, Secure World!'
}
```

CBOR Binary in Hex

```
   A3025873825824822F5820CEDB0457952F7DD0A33FA4692F73BC833A6A6E
   2300B16F6605993F0192E3F219584AD28443A10126A0F6584071E3869E4E
   134A78C95D7ED81F5911FEA4F189EC33C0F6474C866569ED3DF7FB4E0D88
   71367BA3C73612A26C9E3984A4E22CAA4BFBCE84DCAC0539AE87BE9D3D03
   58CEA601010203035884A20281844B544545502D44657669636548536563
   7572654653508D82573A926D4754935332DC29997F744274610458548614
   A40150C0DDD5F15243566087DB4F5B0AA26C2F0250DB42F7093D8C55BAA8
   C5265FC5820F4E035824822F58208CF71AC86AF31BE184EC7A05A411A8C3
   A14FD9B77A30D046397481469468ECE80E14010F020F05844B544545502D
   4465766963654853656375726546535 08D82573A926D4754935332DC2999
   7F744473756974144C8614A11563237463150F030F1818448218210F6323
   74635448656C6C6F2C2053656375726520576F726C6421
```

Example 3: Supplying Personalization Data for Trusted Component Binary

This example uses the following parameters:

* SUIT Profile: suit-sha256-es256-ecdh-a128ctr (see [I-D.ietf-suit-mti] Section 3.2)
  - Algorithm for payload encryption: A128CTR (-65534)
  - Algorithm for key wrap: ECDH-ES + A128KW (-29)
* KEK (Receiver's Private Key):
  - kty: EC2
  - crv: P-256
  - x: h'5886CD61DD875862E5AAA820E7A15274C968A9BC96048DDCACE32F50C3651BA3'
  - y: h'9EED8125E932CD60C0EAD3650D0A485CF726D378D1B016ED4298B2961E258F1B'
  - d: h'60FE6DD6D85D5740A5349B6F91267EEAC5BA81B8CB53EE249E4B4EB102C476B3'
* COSE_KDF_Context
  - AlgorithmID: -3 (A128KW)
  - SuppPubInfo
    - keyDataLength: 128
    - protected: << {/ alg / 1: -29 / ECDH-ES+A128KW / } >>
    - other: 'SUIT Payload Encryption'

CBOR Diagnostic Notation of SUIT Manifest

```
/ SUIT_Envelope / {
  / authentication-wrapper / 2: << [
    << [
      / digest-algorithm-id: / -16 / SHA256 /,
      / digest-bytes: / h'C6E33791C3EA4235D3069E849CCF00390769E0118342161184B293F8893DF010'
    ] >>,
    << / COSE_Sign1_Tagged / 18([
      / protected: / << {
```

```
        / algorithm-id / 1: -7 / ES256 /
      } >>,
      / unprotected: / {},
      / payload: / null,
      / signature: / h'E2F02EB95698DF7D3C9B3B5B0A64AF58B363AD0B3E12AF77C279EBD7B503C9BE48
58C36614919C110E5C294FFB1538EE234CAED278939B7260A4BB63E1970146'
    ]) >>
  ] >>,
  / manifest / 3: << {
    / manifest-version / 1: 1,
    / manifest-sequence-number / 2: 3,
    / common / 3: << {
      / dependencies / 1: {
        / component-index / 1: {
          / dependency-prefix / 1: [
             'TEEP-Device',
             'SecureFS',
            h'8D82573A926D4754935332DC29997F74', / tc-uuid /
             'suit'
          ]
        }
      },
      / components / 2: [
        [
          'TEEP-Device',
          'SecureFS',
          'config.json'
        ]
      ],
      / shared-sequence / 4: << [
        / directive-set-component-index / 12, 0,
        / directive-override-parameters / 20, {
          / parameter-vendor-identifier / 1: h'C0DDD5F15243566087DB4F5B0AA26C2F',
          / parameter-class-identifier / 2: h'DB42F7093D8C55BAA8C5265FC5820F4E'
        },
        / condition-vendor-identifier / 1, 15,
        / condition-class-identifier / 2, 15
      ] >>
    } >>,
    / manifest-component-id / 5: [
      'TEEP-Device',
      'SecureFS',
      'config.suit'
    ],
    / validate / 7: << [
      / directive-set-component-index / 12, 0,
      / directive-override-parameters / 20, {
        / NOTE: image-digest and image-size of plaintext config.json /
        / parameter-image-digest / 3: << [
```

```
          / digest-algorithm-id: / -16 / SHA256 /,
          / digest-bytes: / h'8273468FB64BD84BB04825F8371744D952B751C73A60F455AF681E16772
6F116'
        ] >>,
        / image-size / 14: 61
      },
      / condition-image-match / 3, 15
    ] >>,
    / dependency-resolution / 15: << [
      / directive-set-component-index / 12, 1,
      / directive-override-parameters / 20, {
        / parameter-image-digest / 3: << [
          / algorithm-id / -16 / SHA256 /,
          / digest-bytes / h'B39B52B0B747EA79588C190F567BFC2C8437BA8A73F7EA983182E79F0148
D59B'
        ] >>,
        / parameter-image-size / 14: 389,
        / parameter-uri / 21: "https://example.org/8d82573a-926d-4754-9353-32dc29997f74.s
uit"
      },
      / directive-fetch / 21, 2
    ] >>,
    / install / 20: << [
      / directive-set-component-index / 12, 1,
      / directive-process-dependency / 11, 0,

      / NOTE: fetch encrypted firmware /
      / directive-set-component-index / 12, 0,
      / directive-override-parameters / 20, {
        / NOTE: encrypted payload and encryption-info /
        / parameter-content / 18: h'C43E94F3B51A5DBB76ECFAD44CA7DEFE71D26A36E10054723DDF0
A93CD9B68D9F4B61FCC31CD0CBE30D3FFDF6AB7541BFF1980968A836E17D3BBDE7332',
        / parameter-encryption-info / 19: << 96([
          / protected: / h'',
          / unprotected: / {
            / alg / 1: -65534 / A128CTR /,
            / IV / 5: h'F8FC5E335366171540C1B416ABFDC9A7'
          },
          / payload: / null / detached ciphertext /,
          / recipients: / [
            [
              / protected: / << {
                / alg / 1: -29 / ECDH-ES + A128KW /
              } >>,
              / unprotected: / {
                / ephemeral key / -1: {
                  / kty / 1: 2 / EC2 /,
                  / crv / -1: 1 / P-256 /,
                  / x / -2: h'7AAF18EC7FAB5071B267FA3B8D8FF248A78DAAD9D9B8318EAE8925089F3
C9431',
                  / y / -3: h'84BADF92D62F3804E8DE964ABB21EC6A732B46B2B02DCD2908E6A666C6D
4871B'
                }
              },
```

```
              / payload: / h'F003092CB552689003EB0ACDD081595E6499FF028745DADF'
            ]
          ]
        ]) >>
      },

      / decrypt encrypted firmware /
      / directive-write / 18, 15 / consumes the SUIT_Encryption_Info above /
      / NOTE: decrypted payload would be ''{"name":"FOO Bar","secret":"0123456789abfcdef0
123456789abcd"}'' /
    ] >>,
    / uninstall / 24: << [
      / directive-set-component-index / 12, 1,
      / directive-process-dependency / 11, 0,
      / directive-set-component-index / 12, 0,
      / directive-unlink / 33, 15
    ] >>
  } >>
}
```

CBOR Binary in Hex

```
A2025873825824822F582037522D96C0F9A6B887A21F4B21CDF02767799C
C3A66EAFD5979250CCE11377E2584AD28443A10126A0F6584084A5B76482
0B927C580BF128CC2CA21AE2656F27A6BCE6D63228915CCCCC32DB23C93A
8518A7DB565BD0348F17978474ED7473C4FDED4A2752EEA93B90BE1FF103
590242A801010203035886A301A101A101844B544545502D446576696365
485365637572654653508D82573A926D4754935332DC29997F7444737569
740281834B544545502D4465766963654853656375726546534B636F6E66
69672E6A736F6E04582D880C0014A20150C0DDD5F15243566087DB4F5B0A
A26C2F0250DB42F7093D8C55BAA8C5265FC5820F4E010F020F05834B5445
45502D4465766963654853656375726546534B636F6E6669672E73756974
075831860C0014A2035824822F58208273468FB64BD84BB04825F8371744
D952B751C73A60F455AF681E167726F1160E183D030F0F5872860C0114A3
035824822F5820B39B52B0B747EA79588C190F567BFC2C8437BA8A73F7EA
983182E79F0148D59B0E19018515783D68747470733A2F2F6578616D706C
652E6F72672F38643832353733612D393236642D343735342D393335332D
3332646332393939376637342E737569741502 1458D88A0C010B000C0014
A212583DF137C0755EA5642248EC04F3D24BEF771B5CCD72C56F33F254F4
0A2381DC7C122C5708A99FE87A702A11053EF1BA86CF9A12B7E81AF80147
5959864E6313588AD8608440A20139FFFD0550A0DB218209E3C43E871A81
CF1BEBC9F9F6818344A101381CA120A401022001215820A1A58EA321C7E3
28FE7DE66283DFC3B4081FE5FA7EF90C570FC88693F857EFB6225820AE96
AC0E18D691D5A8066BF95913252F57566F5A07EEF8643822ADD9510ADBBD
58183C09CFE4A33D69F0ADDB73EA728E942791139BA864A9369E120F1818
4A880C010B000C0018210F
```

F. Examples of SUIT Reports

This section shows some examples of SUIT reports.

F.1. Example 1: Success

SUIT Reports have no records if no conditions have failed. The URI in this example is the reference URI provided in the SUIT manifest.

```
{
  / suit-report-manifest-digest / 1:<<[
    / algorithm-id / -16 / "sha256" /,
    / digest-bytes / h'a7fd6593eac32eb4be578278e6540c5c'
                     h'09cfd7d4d234973054833b2b93030609'
  ]>>,
  / suit-report-manifest-uri / 2: "tam.teep.example/personalisation",
  / suit-report-records / 4: []
}
```

F.2. Example 2: Faiure

```
{
  / suit-report-manifest-digest / 1:<<[
    / algorithm-id / -16 / "sha256" /,
    / digest-bytes / h'a7fd6593eac32eb4be578278e6540c5c09cfd7d4d234973054833b2b93030609'
  ]>>,
  / suit-report-manifest-uri / 2: "tam.teep.example/personalisation",
  / suit-report-records / 4: [
    {
      / suit-record-manifest-id / 1:[],
      / suit-record-manifest-section / 2: 7 / dependency-resolution /,
      / suit-record-section-offset / 3: 66,
      / suit-record-dependency-index / 5: 0,
      / suit-record-failure-reason / 6: 404
    }
  ]
}
```

where the dependency-resolution refers to:

```
   {
     authentication-wrapper,
     / manifest / 3:<<{
       / manifest-version / 1:1,
       / manifest-sequence-number / 2:3,
       common,
       dependency-resolution,
       install,
       validate,
       run,
       text
     }>>,
   }
```

and the suit-record-section-offset refers to:

```
   <<[
     / directive-set-dependency-index / 13,0,
     / directive-set-parameters / 19,{
       / uri / 21:'tam.teep.example/'
                 'edd94cd8-9d9c-4cc8-9216-b3ad5a2d5b8a',
       } ,
     / directive-fetch / 21,2,
     / condition-image-match / 3,15
   ]>>,
```

Authors' Addresses

Hannes Tschofenig
Austria
Email: hannes.tschofenig@gmx.net

Mingliang Pei
Broadcom
United States of America
Email: mingliang.pei@broadcom.com

David Wheeler
Amazon
United States of America
Email: davewhee@amazon.com

Dave Thaler
Microsoft
United States of America

Email: dave.thaler.ietf@gmail.com

Akira Tsukamoto
Openchip & Software Technologies, S.L.
Spain
Email: akira.tsukamoto@gmail.com