Workgroup: TEEP
Internet-Draft: draft-ietf-teep-protocol-11
Published: 24 October 2022
Intended Status: Standards Track
Expires: 27 April 2023
Authors: H. Tschofenig
Arm Ltd.
M. Pei
Broadcom
D. Wheeler
Amazon
D. Thaler
Microsoft
A. Tsukamoto
AIST

Trusted Execution Environment Provisioning (TEEP) Protocol

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 27 April 2023.

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 [I-D.ietf-teep-architecture] 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 [I-D.ietf-teep-architecture].

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).

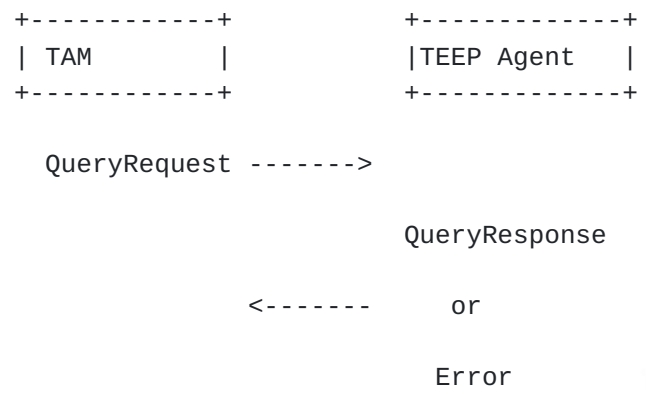
Attestation related terms, such as Evidence and Attestation Results, are as defined in [I-D.ietf-rats-architecture].

3. Message Overview

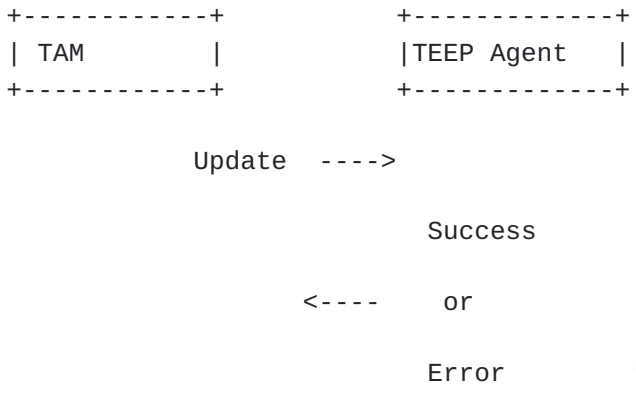
The TEEP protocol consists of messages exchanged between a TAM and a TEEP Agent. The messages are encoded in CBOR and designed to provide end-to-end security. TEEP protocol messages are signed by the endpoints, i.e., the TAM and the TEEP Agent, but Trusted Applications may also be encrypted and signed by a Trusted Component Developer or Device Administrator. The TEEP protocol not only uses CBOR but also the respective security wrapper, namely COSE [RFC8152]. 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 structure. 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, uint (0..23)
$teep-type = uint .size 1
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 [RFC8152] specification.

3. Create a COSE_Sign1 object using the TEEP message as the COSE_Sign1 Payload; all steps specified in [RFC8152] for creating a COSE_Sign1 object MUST be followed.

4.1.2. Validating a TEEP Message

When TEEP message is received (see the ProcessTeepMessage conceptual API defined in [I-D.ietf-teep-architecture] section 6.2.1), 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 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 [RFC8152] ("Signing Objects") for validating a COSE_Sign1 object. The COSE_Sign1 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 via the request parameter. Currently, the following features are supported:

*Request for attestation information,

*Listing supported extensions,

*Querying installed Trusted Components, and

*Listing supported SUIT commands.

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 ],
    * $$query-request-extensions,
    * $$teep-option-extensions
  },
  supported-cipher-suites: [ + $cipher-suite ],
  data-item-requested: uint .bits data-item-requested
]
```

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. The size of the token is at least 8 bytes (64 bits) and maximum of 64 bytes, which is the same as in an EAT Nonce Claim (see [I-D.ietf-rats-eat] Section 3.3). 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-cipher-suites
The supported-cipher-suites parameter lists the cipher suites supported by the TAM. Details about the cipher suite encoding can be found in Section 8.

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 the attestation payload returned with a QueryResponse message. It MUST be absent if the attestation bit is clear (since the token is used instead in that case). When a challenge is provided in the QueryRequest and an EAT is returned with a QueryResponse message then the challenge contained in this request MUST be used to generate the EAT, such as by copying the challenge into the nonce claim found in the EAT if using the Nonce freshness mechanism. 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 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.

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-cipher-suite => $cipher-suite,
    ? selected-version => version,
    ? attestation-payload-format => text,
    ? attestation-payload => bstr,
    ? suit-reports => [ + SUIT_Report ],
    ? tc-list => [ + system-property-claims ],
    ? requested-tc-list => [ + requested-tc-info ],
    ? unneeded-manifest-list => [ + bstr .cbor SUIT_Digest ],
    ? ext-list => [ + ext-info ],
    * $$query-response-extensions,
    * $$teep-option-extensions
  }
]

requested-tc-info = {
  component-id => SUIT_Component_Identifier,
  ? tc-manifest-sequence-number => .within 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-cipher-suite
: The selected-cipher-suite parameter indicates the selected cipher suite. If this parameter is not present, it is to be treated as if the TEEP Agent accepts any cipher suites listed in the QueryRequest, so the TAM can select one. Details about the cipher suite encoding can be found in Section 8.

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=https://datatracker.ietf.org/doc/html/draft-ietf-teep-protocol-10" (see [I-D.lundblade-rats-eat-media-type] for further discussion). (RFC-editor: upon RFC publication, replace URI above with "https://www.rfc-editor.org/info/rfcXXXX" where XXXX is 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 in Section 4 of [I-D.ietf-suit-report]. 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 [I-D.ietf-teep-architecture] 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 Digest. A TEEP Agent can get this information from the UnrequestTA conceptual API defined in [I-D.ietf-teep-architecture] 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 [I-D.ietf-teep-architecture] 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 [I-D.ietf-teep-architecture] 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	[I-D.ietf-rats-eat] section 4.1
Device unique identifier	ueid	[I-D.ietf-rats-eat] section 4.2.1
Vendor of the device	oemid	[I-D.ietf-rats-eat] section 4.2.3
Class of the device	hardware-model	[I-D.ietf-rats-eat] section 4.2.4

Requirement	Claim	Reference
TEE hardware type	hardware-version	[I-D.ietf-rats-eat] section 4.2.5
TEE hardware version	hardware-version	[I-D.ietf-rats-eat] section 4.2.5
TEE firmware type	manifests	[I-D.ietf-rats-eat] section 4.2.16
TEE firmware version	manifests	[I-D.ietf-rats-eat] section 4.2.16

Table 1

The "manifests" claim 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 [I-D.ietf-rats-architecture]) that can be presented to other Relying Parties.

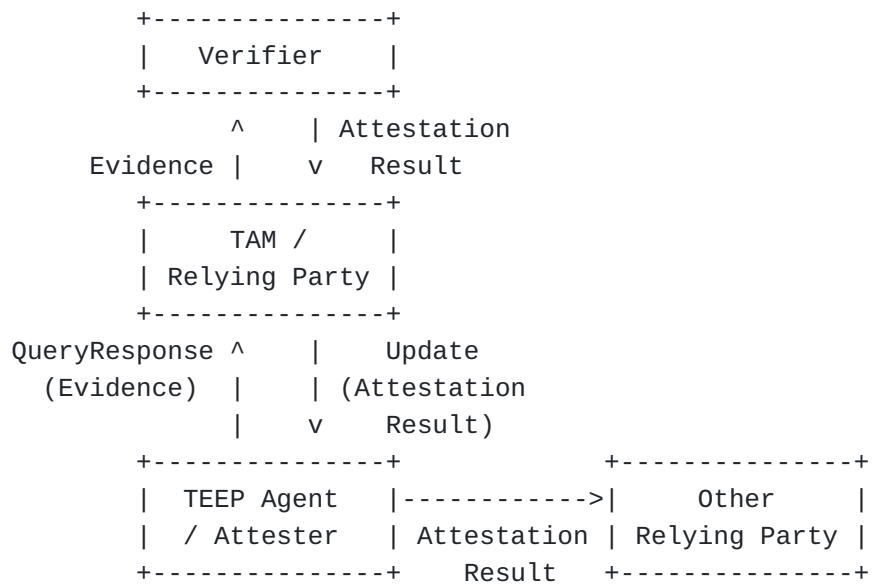


Figure 1: Example use of TEEP and attestation

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 => [ + bstr .cbor SUIT_Digest ],
    ? manifest-list => [ + bstr .cbor SUIT_Envelope ],
    ? attestation-payload-format => text,
    ? attestation-payload => bstr,
    * $$update-extensions,
    * $$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 Digest.

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=https://datatracker.ietf.org/doc/html/draft-ietf-teep-protocol-10" (see [I-D.lundblade-rats-eat-media-type] for further discussion). (RFC-editor: upon RFC publication, replace URI above with "https://www.rfc-editor.org/info/rfcXXXX" where XXXX is 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.

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 [I-D.ietf-teep-architecture] 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.suit" that contains the URI. The filename "tc-uuid.suit" 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.

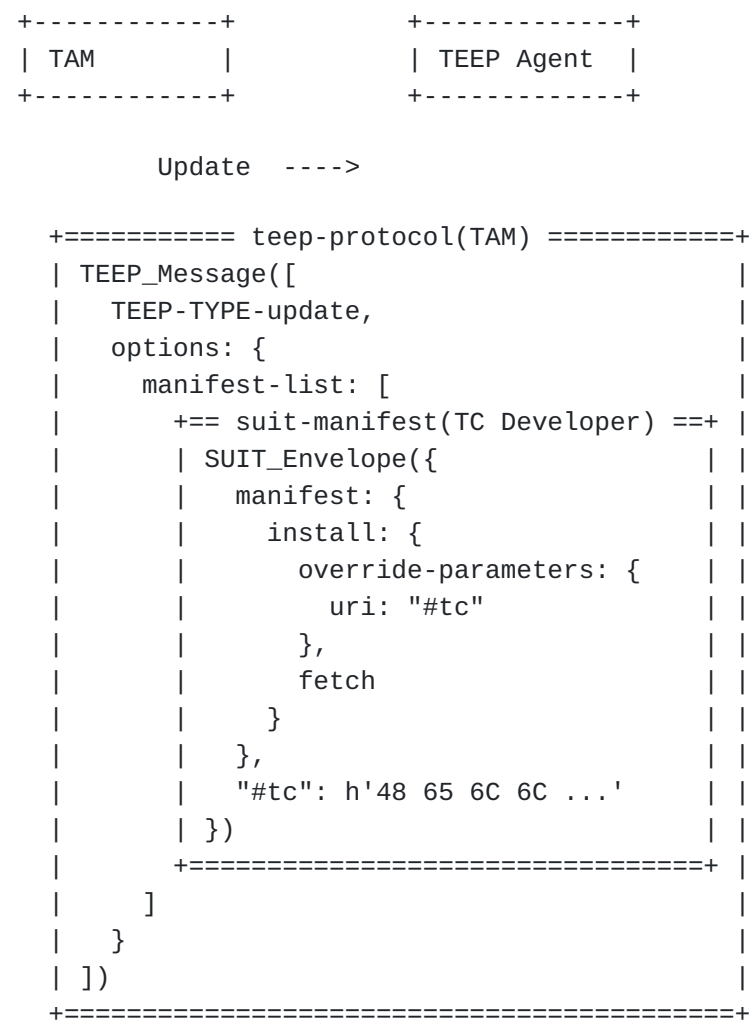


Figure 3: Integrated Payload with Trusted Component Binary

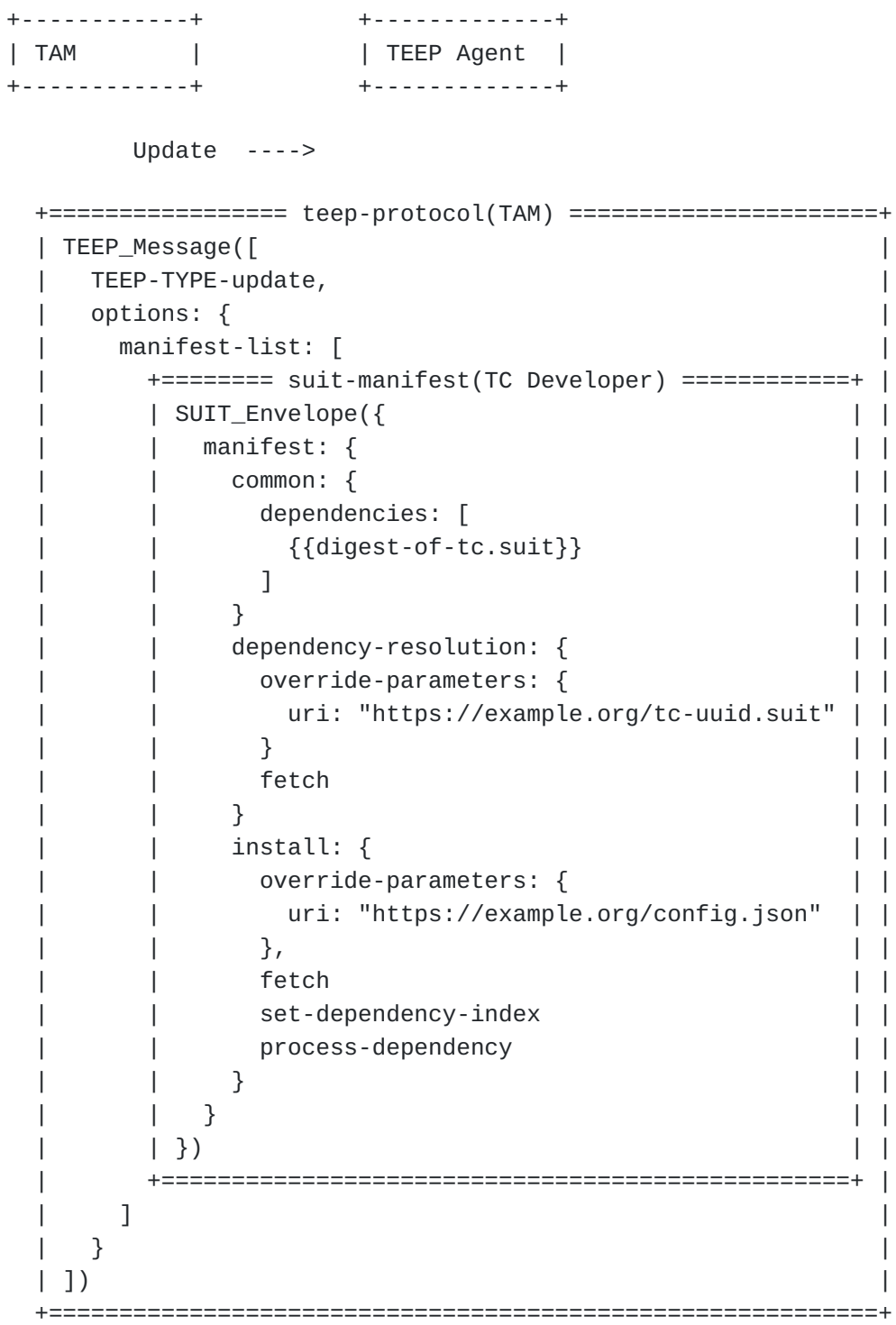
For the full SUIT Manifest example binary, see Appendix "Example 2: SUIT Manifest including the Trusted Component Binary".

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.suit" from Scenario 1.

The Trusted Component Developer places Personalization Data in a file named "config.json" and hosts it on an HTTPS server. The Trusted Component Developer then creates a SUIT manifest with the URI, specifying which Trusted Component Binary it correlates to in the parameter 'dependency-resolution', and signs the SUIT manifest.

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: [	
	{{digest-of-tc.suit}}	
	]	
	}	
	dependency-resolution: {	
	override-parameters: {	
	uri: "https://example.org/tc-uuid.suit"	
	}	
	fetch	
	}	
	install: {	
	override-parameters: {	
	uri: "https://example.org/config.json"	
	},	
	fetch	
	set-dependency-index	
	process-dependency	
	}	
	}	
	})	
+==+
]
}
])
+==+
and then,
+-------------+ +--------------+
| TEEP Agent | | TC Developer |
+-------------+ +--------------+
<----
fetch "https://example.org/config.json"

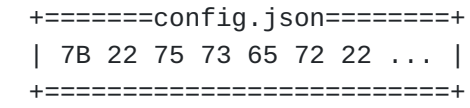


Figure 4: Personalization Data

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

4.4.4. Scenario 4: Unlinking a Trusted Component

A Trusted Component Developer can also generate a SUIT Manifest that unlinks the installed Trusted Component. The TAM delivers it when the TAM wants to uninstall the component.

The suit-directive-unlink (see [I-D.ietf-suit-trust-domains] Section-6.5.4) is located in the manifest to unlink the Trusted Component, meaning that the reference count is decremented and the component is deleted when the reference count becomes zero. (If other Trusted Components depend on it, the reference count will not be zero.)

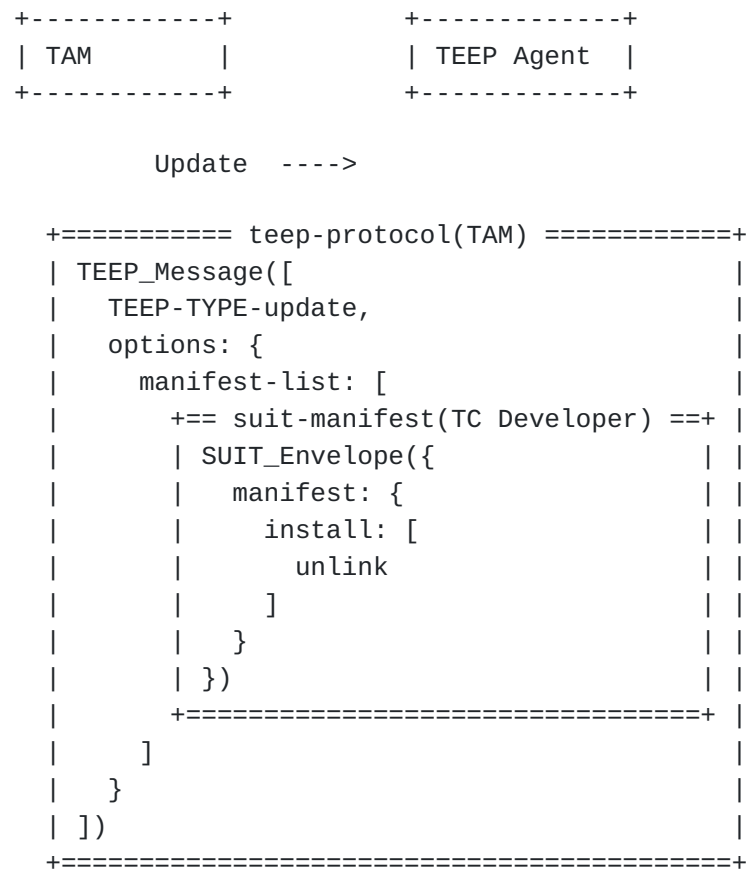


Figure 5: Unlink Trusted Component example (summary)

For the full SUIT Manifest example binary, see Appendix E. SUIT Example 4 (Appendix "E.4. Example 4: Unlink a Trusted Component")

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-cipher-suites => [ + $cipher-suite ],
     ? supported-freshness-mechanisms => [ + $freshness-mechanism ],
     ? versions => [ + version ],
     ? suit-reports => [ + SUIT_Report ],
     * $$teep-error-extensions,
     * $$teep-option-extensions
  },
  err-code: 0..23
]
```

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-cipher-suites
: The supported-cipher-suites parameter lists the cipher suite(s) supported by the TEEP Agent. Details about the cipher suite encoding can be found in Section 8. 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.

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 TEEP request 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. A TAM receiving this error might refuse to communicate further with the TEEP Agent 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 Agent does not support an extension included in the request message. For diagnosis purposes it is RECOMMMENDED to identify the unsupported extension in the error message. A TAM receiving this error might retry the request without using 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 Agent does not support the TEEP protocol version indicated in the request 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. For example, the certificate was of an unsupported type, or the certificate was revoked by its signer. A TAM receiving this error might attempt to use an alternate certificate.

ERR_CERTIFICATE_EXPIRED (9)
A certificate has expired or is not currently valid. A TAM 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 request message. A TAM receiving this error might retry the same request at a later point in time.

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

https://datatracker.ietf.org/doc/html/draft-ietf-teep-protocol-10. (RFC-editor: upon RFC publication, replace string with "https://www.rfc-editor.org/info/rfcXXXX" where XXXX is 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.

*COSE/JOSE Protection: See Section 8.

*Detached EAT Bundle Support: DEB use is permitted.

*Verification 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).

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

*Freshness: See Section 9.

*Required Claims: None.

*Prohibited Claims: None.

*Additional Claims: Optional claims are those listed in Section 4.3.1.

*Refined Claim Definition: None.

*CBOR Tags: CBOR Tags are not used.

*Manifests and Software Evidence Claims: The sw-name claim for a Trusted Component holds the URI of the SUIT manifest for that component.

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.

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-cipher-suites	1
challenge	2
versions	3
selected-cipher-suite	5
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

Name	Label
component-id	16
tc-manifest-sequence-number	17
have-binary	18
suit-reports	19
token	20
supported-freshness-mechanisms	21

Table 2

7. Behavior Specification

Behavior is specified in terms of the conceptual APIs defined in section 6.2.1 of [I-D.ietf-teep-architecture].

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. 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 send in the QueryRequest message if a token was present. If these requirements are not met, the TAM drops the message. 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 [I-D.ietf-rats-architecture]) 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

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 can 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.

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 QueryRequest 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 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.

When a QueryRequest message is received, the Agent responds with a QueryResponse message if all fields were understood, or an Error message if any error was encountered.

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

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.

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

; The following two cipher suites have only a single operation each.
; Other cipher suites may be defined to have multiple operations.

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

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

cose-sign1 = 18      ; CoAP Content-Format value

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 [RFC8152] 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 [I-D.ietf-teep-architecture] 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 [I-D.ietf-teep-architecture].

The cipher suites defined above do not do encryption at the TEEP layer, but permit encryption of the SUIT payload (e.g., using [I-D.ietf-suit-firmware-encryption]). See Section 10 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-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 one of the cipher suites mandatory for the TAM.

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

9. 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 [I-D.ietf-rats-architecture].

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_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. 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.

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 digitial signatures and can support symmetric 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 signed and encrypted (e.g.,

via [I-D.ietf-suit-firmware-encryption]) by a Trusted Component Signer other than the TAM is opaque to the TAM.

TEEP Broker

As discussed in section 6 of [I-D.ietf-teep-architecture], 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.

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.wallace-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 using a firmware update mechanism such as [I-D.wallace-rats-concise-ta-stores]. If the CA issuing certificates to devices gets compromised then these devices might 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.

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.

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 [RFC7049].

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-rats-architecture] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-22, 28 September 2022, <https://www.ietf.org/archive/id/draft-ietf-rats-architecture-22.txt>.

[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-17, 22 October 2022, <https://www.ietf.org/archive/id/draft-ietf-rats-eat-17.txt>.

[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-06, 7 September 2022, <https://www.ietf.org/archive/id/draft-ietf-rats-reference-interaction-models-06.txt>.

[I-D.ietf-suit-manifest] Moran, B., Tschofenig, H., Birkholz, H., Zandberg, K., and O. Rønningstad, "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-20, 7 October 2022, <https://www.ietf.org/archive/id/draft-ietf-suit-manifest-20.txt>.

[I-D.ietf-suit-report]
Moran, B. and H. Birkholz, "Secure Reporting of Update Status", Work in Progress, Internet-Draft, draft-ietf-suit-report-04, 24 October 2022, <https://www.ietf.org/archive/id/draft-ietf-suit-report-04.txt>.

[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-01, 24 October 2022, <https://www.ietf.org/archive/id/draft-ietf-suit-trust-domains-01.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[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/info/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/info/rfc5198>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

13.2. Informative References

[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-09, 24 October 2022, <https://www.ietf.org/archive/id/draft-ietf-suit-firmware-encryption-09.txt>.

[I-D.ietf-teep-architecture] Pei, M., Tschofenig, H., Thaler, D., and D. M. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-ietf-teep-architecture-19, 24 October 2022,

<https://www.ietf.org/archive/id/draft-ietf-teep-architecture-19.txt>.

[I-D.lundblade-rats-eat-media-type] Lundblade, L., Birkholz, H., and T. Fossati, "EAT Media Types", Work in Progress, Internet-Draft, draft-lundblade-rats-eat-media-type-00, 26 May 2022, <https://www.ietf.org/archive/id/draft-lundblade-rats-eat-media-type-00.txt>.

[I-D.wallace-rats-concise-ta-stores] Wallace, C., Housley, R., Fossati, T., and Y. Deshpande, "Concise TA Stores (CoTS)", Work in Progress, Internet-Draft, draft-wallace-rats-concise-ta-stores-01, 10 October 2022, <https://www.ietf.org/archive/id/draft-wallace-rats-concise-ta-stores-01.txt>.

[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/info/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/info/rfc8915>.

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.

```
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, uint (0..23)
$teep-type = uint .size 1
TEEP-TYPE-query-request = 1
TEEP-TYPE-query-response = 2
TEEP-TYPE-update = 3
TEEP-TYPE-teep-success = 5
TEEP-TYPE-teep-error = 6

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,
)

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 ],
    * $$query-request-extensions,
    * $$teep-option-extensions
  },
  supported-cipher-suites: [ + $cipher-suite ],
  data-item-requested: uint .bits data-item-requested
]
```

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

;MANDATORY for TAM to support the following cipher-suites, 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 ]

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

; freshness-mechanisms

FRESHNESS_NONCE = 0
FRESHNESS_TIMESTAMP = 1
FRESHNESS_EPOCH_ID = 2

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

query-response = [
  type: TEEP-TYPE-query-response,
  options: {
    ? token => bstr .size (8..64),
    ? selected-cipher-suite => $cipher-suite,
    ? selected-version => version,
    ? attestation-payload-format => text,
    ? attestation-payload => bstr,
    ? suit-reports => [ + SUIT_Report ],
    ? tc-list => [ + system-property-claims ],
    ? requested-tc-list => [ + requested-tc-info ],
    ? unneeded-manifest-list => [ + bstr .cbor SUIT_Digest ],
    ? 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 => [ + bstr .cbor SUIT_Digest ],
    ? manifest-list => [ + bstr .cbor SUIT_Envelope ],
    * $$update-extensions,
    * $$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-cipher-suites => [ + $cipher-suite ],
     ? supported-freshness-mechanisms => [ + $freshness-mechanism ],
     ? 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_CERTIFICATE_EXPIRED = 9
ERR_TEMPORARY_ERROR = 10
ERR_MANIFEST_PROCESSING_FAILED = 17

; labels of mapkey for teep message parameters, uint (0..23)
supported-cipher-suites = 1
challenge = 2
versions = 3
selected-cipher-suite = 5
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
```

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-cipher-suites: / [ [ [ 18, -7 ] ], / Sign1 using ES256 /
                                 [ [ 18, -8 ] ]  / Sign1 using EdDSA /
                               ],
  / data-item-requested: / 3 / attestation | trusted-components /
]
```

D.1.2. CBOR Binary Representation

```
84                 # array(4)
   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-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 /
   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 = /
{
    / issuer /                   1: "joe",
    / timestamp (iat) /          6: 1(1526542894)
    / nonce /                   10: h'948f8860d13a463e8e',
    / secure-boot /             15: true,
    / debug-status /            16: 3, / disabled-permanently /
    / security-level /          14: 3, / secure-restricted /
    / device-identifier /    <TBD>: h'e99600dd921649798b013e9752dcf0c5',
    / vendor-identifier /    <TBD>: h'2b03879b33434a7ca682b8af84c19fd4',
    / class-identifier /     <TBD>: h'9714a5796bd245a3a4ab4f977cb8487f',
    / chip-version /            26: [ "MyTEE", 1 ],
    / component-identifier / <TBD>: h'60822887d35e43d5b603d18bcaa3f08d',
    / version /              <TBD>: "v0.1"
}
```

D.3. QueryResponse Message

D.3.1. CBOR Diagnostic Notation

```
/ query-response = /
[
  / type: / 2 / TEEP-TYPE-query-response /,
  / options: /
  {
    / token / 20 : h'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF',
    / selected-cipher-suite / 5 : [ [ 18, -7 ] ] / only use ES256 /,
    / selected-version / 6 : 0,
    / attestation-payload / 7 : h'' / empty only for example purpose /,
    / tc-list / 8 : [
      {
        / system-component-id / 0 : [ h'0102030405060708090A0B0C0D0E0F'
      },
      {
        / system-component-id / 0 : [ h'1102030405060708090A0B0C0D0E0F'
      }
    ]
  }
]
```

D.3.2. CBOR Binary Representation

```
82                  # array(2)
   02               # unsigned(2) / TEEP-TYPE-query-response /
   A5               # map(5)
      14            # unsigned(20) / token: /
      50            # bytes(16)
         A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
      05            # unsigned(5) / selected-cipher-suite: /
      81            # array(1)
         82         # array(2)
            12      # unsigned(18) / cose-sign1 /
            26      # negative(6) / -7 = cose-alg-es256 /
      06            # unsigned(6) / selected-version: /
      00            # unsigned(0)
      07            # unsigned(7) / attestation-payload: /
      40            # bytes(0)
                    # ""
      08            # unsigned(8) / tc-list: /
      82            # array(2)
         A1         # map(1)
            00      # unsigned(0) / system-component-id: /
            81      # array(1)
               4F   # bytes(15)
                  0102030405060708090A0B0C0D0E0F
         A1         # map(1)
            00      # unsigned(0) / system-component-id: /
            81      # array(1)
               4F   # bytes(15)
                  1102030405060708090A0B0C0D0E0F
```

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'DB601ADE73092B58532CA03FBB663DE49
            ] >>,
            << / COSE_Sign1_Tagged / 18( [
              / protected: / << {
                / algorithm-id / 1: -7 / ES256 /
              } >>,
              / unprotected: / {},
              / payload: / null,
              / signature: / h'5B2D535A2B6D5E3C585C1074F414DA9E10BD285C9
            ] ) >>
          ] >>,
          / 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'C0DDD5F15243
                  / suit-parameter-class-identifier / 2: h'DB42F7093D8C5
                  / suit-parameter-image-digest / 3: << [
                    / suit-digest-algorithm-id: / -16 / suit-cose-alg-sh
                    / suit-digest-bytes: / h'8CF71AC86AF31BE184EC7A05A41
                  ] >>,
                  / 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
              },
              / 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
            6563757265465350 8D82573A926D4754935332DC29997F74427461045854
            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 / {
  / suit-authentication-wrapper / 2: << [
    << [
      / suit-digest-algorithm-id: / -16 / suit-cose-alg-sha256 /,
      / suit-digest-bytes: / h'DB601ADE73092B58532CA03FBB663DE4953243533
    ] >>,
    << / COSE_Sign1_Tagged / 18( [
      / protected: / << {
        / algorithm-id / 1: -7 / ES256 /
      } >>,
      / unprotected: / {},
      / payload: / null,
      / signature: / h'5B2D535A2B6D5E3C585C1074F414DA9E10BD285C99A33916D
    ] ) >>
  ] >>,
  / 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'C0DDD5F15243566087DB
          / suit-parameter-class-identifier / 2: h'DB42F7093D8C55BAA8C52
          / suit-parameter-image-digest / 3: << [
            / suit-digest-algorithm-id: / -16 / suit-cose-alg-sha256 /,
            / suit-digest-bytes: / h'8CF71AC86AF31BE184EC7A05A411A8C3A14
          ] >>,
          / 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-47
      },
      / suit-directive-fetch / 21, 15,
      / suit-condition-image-match / 3, 15
    ] >>
```

```
  } >>
}
```

CBOR Binary in Hex

```
A2025873825824822F5820DB601ADE73092B58532CA03FBB663DE495
32435336F1558B49BB622726A2FEDD584AD28443A10126A0F658405B2D53
5A2B6D5E3C585C1074F414DA9E10BD285C99A33916DADE3ED38812504817
AC48B62B8E984EC622785BD1C411888BE531B1B594507816B201F6F28579
A40358D4A401010203035884A20281844B544545502D4465766963654853
6563757265465350 8D82573A926D4754935332DC29997F74427461045854
8614A40150C0DDD5F15243566087DB4F5B0AA26C2F0250DB42F7093D8C55
BAA8C5265FC5820F4E035824822F58208CF71AC86AF31BE184EC7A05A411
A8C3A14FD9B77A30D046397481469468ECE80E14010F020F0958458614A1
15783B68747470733A2F2F6578616D706C652E6F72672F38643832353733
612D393236642D343735342D393335332D333264633239393937663742E
7461150F030F
```

Example 2: SUIT Manifest including the Trusted Component Binary

CBOR Diagnostic Notation of SUIT Manifest

```
/ SUIT_Envelope / {
  / suit-authentication-wrapper / 2: << [
    / digest: / << [
      / suit-digest-algorithm-id: / -16 / SHA-256 /,
      / suit-digest-bytes: / h'E8B5EC4510260B42B489FDEC4B4918E8E97EB6E13
    ] >>,
    / signatures: / << / COSE_Sign1_Tagged / 18( [
      / protected: / << {
        / alg / 1: -7 / ES256 /
      } >>,
      / unprotected: / {},
      / payload: / null,
      / signature: / h'C3C646030A93EC39E3F27111BE73A2810A9F7A57BB34E9C99
    ]) >>
  ] >>,
  / manifest / 3: << {
    / manifest-version / 1: 1,
    / manifest-sequence-number / 2: 3,
    / common / 3: << {
      / components / 2: [
        [
          h'544545502D446576696365',            / "TEEP-Device" /
          h'5365637572654653',                  / "SecureFS" /
          h'8D82573A926D4754935332DC29997F74',  / tc-uuid /
          h'7461'                               / "ta" /
        ]
      ],
      / common-sequence / 4: << [
        / directive-override-parameters / 20, {
          / vendor-id / 1: h'C0DDD5F15243566087DB4F5B0AA26C2F' / c0ddd5f
          / class-id / 2: h'DB42F7093D8C55BAA8C5265FC5820F4E' / db42f709
          / image-digest / 3: << [
            / algorithm-id: / -16 / SHA-256 /,
            / digest-bytes: / h'8CF71AC86AF31BE184EC7A05A411A8C3A14FD9B7
          ] >>,
          / image-size / 14: 20
        },
        / condition-vendor-identifier / 1, 15,
        / condition-class-identifier / 2, 15
      ] >>
    } >>,
    / install / 17: << [
      / directive-override-parameters / 20, {
        / uri / 21: "#tc"
      },
      / directive-fetch / 21, 15,
      / condition-image-match / 3, 15
    ] >>
  } >>,
```

```
  "#tc" : h'48656C6C6F2C20536563757265205365637572652C20576F726C6421' / "Hello, Secure W
}
```

CBOR Binary in Hex

```
A3025873825824822F5820E8B5EC4510260B42B489FDEC4B4918E8E9
7EB6E135C1B3B40E82419BF79224DE584AD28443A10126A0F65840C3C646
030A93EC39E3F27111BE73A2810A9F7A57BB34E9C9916FC0601EAB8EB506
B96C70864149664C1D090757714ACE153FBB982DFDA5B3FC150D89581E39
9403589AA401010203035884A20281844B544545502D4465766963654853
65637572654653508D82573A926D4754935332DC29997F74427461045854
8614A40150C0DDD5F15243566087DB4F5B0AA26C2F0250DB42F7093D8C55
BAA8C5265FC5820F4E035824822F58208CF71AC86AF31BE184EC7A05A411
A8C3A14FD9B77A30D046397481469468ECE80E14010F020F114C8614A115
63237463150F030F632374635448656C6C6F2C2053656375726520576F72
6C6421
```

Example 3: Supplying Personalization Data for Trusted Component Binary

CBOR Diagnostic Notation of SUIT Manifest

```
/ SUIT_Envelope / {
  / authentication-wrapper / 2: << [
    / digest: / << [
      / algorithm-id: / -16 / SHA-256 /,
      / digest-bytes: / h'B2967C80D2DA2C9C226331AC4CF4C147F1D9E059C4EB6D
    ] >>,
    / signatures: / << 18([
      / protected: / << {
        / alg / 1: -7 / ES256 /
      } >>,
      / unprotected: / {
      },
      / payload: / null,
      / signature: / h'BE370C83AAF922A2D2A807D068879EE3D1F1781750181EEE0
    ]) >>
  ] >>,
  / manifest / 3: << {
    / manifest-version / 1: 1,
    / manifest-sequence-number / 2: 3,
    / common / 3: << {
      / dependencies / 1: [
        / dependency-digest / 1: [
          / algorithm-id: / -16 / SHA-256 /,
          / digest-bytes: / h'549B1BF2E6F662167342A91E2CD16A695BE2ECFB7C
        ]
      ],
      / components / 2: [
        [
          h'544545502D446576696365',  / "TEEP-Device" /
          h'5365637572654653',        / "SecureFS" /
          h'636F6E6669672E6A736F6E'   / "config.json" /
        ]
      ],
      / common-sequence / 4: << [
        / directive-set-component-index / 12, 0,
        / directive-override-parameters / 20, {
          / vendor-id / 1: h'C0DDD5F15243566087DB4F5B0AA26C2F' / c0ddd5f
          / class-id / 2: h'DB42F7093D8C55BAA8C5265FC5820F4E' / db42f709
          / image-digest / 3: << [
            / algorithm-id: / -16 / SHA-256 /,
            / digest-bytes: / h'AAABCCCDEEEF000122234445666678889ABBBCDDD
          ] >>,
          / image-size / 14: 64
        },
        / condition-vendor-identifier / 1, 15,
        / condition-class-identifier / 2, 15
      ] >>
    } >>,
    / validate / 7: << [
```

```
      / directive-set-component-index / 12, 0,
      / condition-image-match / 3, 15
    ] >>,
    / dependency-resolution / 15: << [
      / directive-set-dependency-index / 13, 0,
      / directive-override-parameters / 20, {
        / uri / 21: "https://example.org/8d82573a-926d-4754-9353-32dc299
      },
      / directive-fetch / 21, 2,
      / condition-image-match / 3, 15
    ] >>,
    / install / 17: << [
      / directive-set-dependency-index / 13, 0,
      / directive-process-dependency / 18, 0,
      / directive-set-component-index / 12, 0,
      / directive-override-parameters / 20, {
        / uri / 21: "https://example.org/config.json"
      },
      / directive-fetch / 21, 2,
      / condition-image-match / 3, 15
    ] >>
  } >>
}
```

CBOR Binary in Hex

```
A2025873825824822F5820B2967C80D2DA2C9C226331AC4CF4C147F1
D9E059C4EB6D165AB43E4C86275B9C584AD28443A10126A0F65840BE370C
83AAF922A2D2A807D068879EE3D1F1781750181EEE0251E96D320356B6E6
D9553B9E33E4D250C52BCD446272F22A00AF6F3C43DAA7F263EF375307F6
4603590134A6010102030358A7A30181A101822F5820549B1BF2E6F66216
7342A91E2CD16A695BE2ECFB7C325639189D0EA8EBA57D0A0281834B5445
45502D446576696365485365637572654653 4B636F6E6669672E6A736F6E
045857880C0014A40150C0DDD5F15243566087DB4F5B0AA26C2F0250DB42
F7093D8C55BAA8C5265FC5820F4E035824822F5820AAABCCCDEEEF000122
23444566678889ABBBCDDDEFFF011123334555677789990E1840010F020F
0745840C00030F0F5849880D0014A115783D68747470733A2F2F6578616D
706C652E6F72672F38643832353733612D393236642D343735342D393335
332D333264633239393937663742E737569741502030F11582F8C0D0012
000C0014A115781F68747470733A2F2F6578616D706C652E6F72672F636F
6E6669672E6A736F6E1502030F
```

E.4. Example 4: Unlink a Trusted Component

CBOR Diagnostic Notation of SUIT Manifest

```
/ SUIT_Envelope / {
  / authentication-wrapper / 2: << [
    / digest: / << [
      / algorithm-id: / -16 / SHA-256 /,
      / digest-bytes: / h'54EA3D80AAF5370527E8C4FC9E0D91FF0BD0FED26AEAB6
    ] >>,
    / signatures: / << / COSE_Sign1_Tagged / 18( [
      / protected: / << {
        / alg / 1: -7 / ES256 /
      } >>,
      / unprotected: / {
      },
      / payload: / null,
      / signature: / h'436A36C33A3300D13ACF0075BA751B419FE1E8CCAB6CFB795
    ]) >>
  ] >>,
  / manifest / 3: << {
    / manifest-version / 1: 1,
    / manifest-sequence-number / 2: 18446744073709551615 / UINT64_MAX /,
    / common / 3: << {
      / components / 2: [
        [
          h'544545502D446576696365',            / "TEEP-Device" /
          h'5365637572654653',                  / "SecureFS" /
          h'8D82573A926D4754935332DC29997F74',  / tc-uuid /
          h'7461'                               / "ta" /
        ]
      ],
      / common-sequence / 4: << [
        / directive-override-parameters / 20, {
          / vendor-id / 1: h'C0DDD5F15243566087DB4F5B0AA26C2F' / c0ddd5f
          / class-id / 2: h'DB42F7093D8C55BAA8C5265FC5820F4E' / db42f709
        },
        / condition-vendor-identifier / 1, 15,
        / condition-class-identifier / 2, 15
      ] >>
    } >>,
    / install / 17: << [
      / directive-set-component-index / 12, 0,
      / directive-unlink / 33, 0
    ] >>
  } >>
}
```

CBOR Binary in Hex

```
A2025873825824822F582054EA3D80AAF5370527E8C4FC9E0D91FF0B
D0FED26AEAB602CA516541FEF7F15A584AD28443A10126A0F65840436A36
C33A3300D13ACF0075BA751B419FE1E8CCAB6CFB7952C2E97FD5DA70278E
A3D8A8377D247CF8FE7F2874DF5A0F31B042C659A98DD57A0DC23F094666
E8035873A40101021BFFFFFFFFFFFFFFFF03585BA20281844B544545502D
4465766963654853656375726546535 08D82573A926D4754935332DC2999
7F7442746104582B8614A20150C0DDD5F15243566087DB4F5B0AA26C2F02
50DB42F7093D8C55BAA8C5265FC5820F4E010F020F1146840C00182100
```

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
  / suit-report-records / 4: []
}
```

F.2. Example 2: Faiure

```
{
  / suit-report-manifest-digest / 1:<<[
    / algorithm-id / -16 / "sha256" /,
    / digest-bytes / h'a7fd6593eac32eb4be578278e6540c5c09cfd7d4d23497305
  ]>>,
  / suit-report-manifest-uri / 2: "tam.teep.example/personalisation.suit
  / 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:

```
107({
  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.suit',
    } ,
  / directive-fetch / 21,2 ,
  / condition-image-match / 3,15
]>>,
```

Authors' Addresses

Hannes Tschofenig
Arm Ltd.
6067 Absam
Austria

Email: hannes.tschofenig@arm.com

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: dthaler@microsoft.com

Akira Tsukamoto
AIST
Japan

Email: akira.tsukamoto@aist.go.jp