TEEP H. Tschofenig
Internet-Draft Arm Ltd.
Intended status: Standards Track M. Pei
Expires: January 13, 2022 Broadcom
D. Wheeler
Amazon
D. Thaler
Microsoft
A. Tsukamoto
AIST
July 12, 2021

Trusted Execution Environment Provisioning (TEEP) Protocol
draft-ietf-teep-protocol-06

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

Copyright Notice

Table of Contents

1. Introduction

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

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

The Trusted Execution Environment Provisioning (TEEP) architecture document [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).

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.

```
  +------------+           +-------------+
  | TAM        |           |TEEP Agent   |
  +------------+           +-------------+

    QueryRequest ------->

                           QueryResponse

                 <-------     or

                             Error
```

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.

```
 +------------+           +-------------+
 | TAM        |           |TEEP Agent   |
 +------------+           +-------------+

             Update  ---->

                            Success

                    <----    or

                            Error
```

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                => (query-request /
                                    query-response /
                                    update /
                                    teep-success /
                                    teep-error ),
}
```

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 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. Prepend the COSE object with the TEEP CBOR tag to indicate that the CBOR-encoded message is indeed a TEEP message.

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. Remove the TEEP message CBOR tag and verify that one of the COSE CBOR tags follows it.

3. Verify that the message contains a COSE_Sign1 structure.

4. 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.

5. 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.

6. 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 ciphersuites, 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-cipher-suites => [ + suite ],
    ? supported-freshness-mechanisms => [ + freshness-mechanism ],
    ? challenge => bstr .size (8..512),
    ? versions => [ + version ],
    ? ocsp-data => bstr,
    * $$query-request-extensions
    * $$teep-option-extensions
  },
  data-item-requested: 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. 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).

data-item-requested
: The data-item-requested parameter indicates what information the TAM requests from the TEEP Agent in the form of a bitmap. Each value in the bitmap corresponds to an IANA registered information element. This specification defines the following initial set of information elements:

  attestation (1) With this value the TAM requests the TEEP Agent to return attestation evidence (e.g., an EAT) 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-commands (8) With this value the TAM queries the TEEP Agent for supported commands offered by the SUIT manifest implementation.

Further values may be added in the future via IANA registration.

supported-cipher-suites
The supported-cipher-suites parameter lists the ciphersuite(s) supported by the TAM. If this parameter is not present, it is to be treated the same as if it contained both ciphersuites defined in this document. Details about the ciphersuite encoding can be found in Section 7.

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 8. If this parameter is absent, it means only the nonce mechanism is supported.

challenge
The challenge field is an optional parameter used for ensuring the freshness of the attestation evidence 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 the QueryResponse message then the challenge contained in this request MUST be copied into the nonce claim found in the EAT. 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.

ocsp-data
The ocsp-data parameter contains a list of OCSP stapling data respectively for the TAM certificate and each of the CA certificates up to, but not including, the trust anchor. The TAM provides OCSP data so that the TEEP Agent can validate the status of the TAM certificate chain without making its own external OCSP service call. OCSP data MUST be conveyed as a DER-encoded OCSP response (using the ASN.1 type OCSPResponse defined in [RFC6960]).

The use of OCSP is OPTIONAL to implement for both the TAM and the TEEP Agent. A TAM can query the TEEP Agent for the support of this functionality via the capability discovery exchange, as described above.

4.3. QueryResponse Message

The QueryResponse message is the successful response by the TEEP Agent after receiving a QueryRequest 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 => suite,
    ? selected-version => version,
    ? evidence-format => text,
    ? evidence => bstr,
    ? tc-list => [ + tc-info ],
    ? requested-tc-list => [ + requested-tc-info ],
    ? unneeded-tc-list => [ + SUIT_Component_Identifier ],
    ? ext-list => [ + ext-info ],
    * $$query-response-extensions,
    * $$teep-option-extensions
  }
]

tc-info = {
  component-id => SUIT_Component_Identifier,
  ? tc-manifest-sequence-number => .within uint .size 8
}

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 ciphersuite. Details about the ciphersuite encoding can be found in Section 7.

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

evidence-format
The evidence-format parameter indicates the IANA Media Type of the attestation evidence contained in the evidence parameter. It MUST be present if the evidence parameter is present and the format is not an EAT.

evidence
The evidence parameter contains the attestation evidence. This parameter MUST be present if the QueryResponse is sent in response to a QueryRequest with the attestation bit set. If the evidence-format parameter is absent, the attestation evidence 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.

tc-list
The tc-list parameter enumerates the Trusted Components installed on the device in the form of tc-info objects. 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 UnrequestTA conceptual API defined in [I-D.ietf-teep-architecture] section 6.2.1.

unneeded-tc-list
The unneeded-tc-list parameter enumerates the Trusted Components that 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 Trusted Component can be deleted. Each unneeded Trusted Component is identified by its SUIT Component Identifier. 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.

The tc-info object has the following fields:

component-id
A SUIT Component Identifier.

tc-manifest-sequence-number
The suit-manifest-sequence-number value from the SUIT manifest for the Trusted Component, if a SUIT manifest was used.

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

Section 7.1 of [I-D.ietf-teep-architecture] lists information that may be required in the evidence depend on the circumstance. When an Entity Attestation Token is used, the following claims can be used to meet those requirements:

Requirement	Claim	Reference
Device unique identifier	device-identifier	[I-D.birkholz-rats-suit-claims] section 3.1.3
Vendor of the device	vendor-identifier	[I-D.birkholz-rats-suit-claims] section 3.1.1
Class of the device	class-identifier	[I-D.birkholz-rats-suit-claims] section 3.1.2
TEE hardware type	chip-version-scheme	[I-D.ietf-rats-eat] section 3.7
TEE hardware version	chip-version-scheme	[I-D.ietf-rats-eat] section 3.7
TEE firmware type	component-identifier	[I-D.birkholz-rats-suit-claims] section 3.1.4
TEE firmware version	version	[I-D.birkholz-rats-suit-claims] section 3.1.8
Freshness proof	nonce	[I-D.ietf-rats-eat] section 3.3

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.

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),
    ? manifest-list => [ + bstr .cbor SUIT_Envelope ],
    * $$update-extensions,
    * $$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.

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.

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.

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.moran-suit-report]. If the suit-report-nonce field is present in the SUIT Report, is value MUST match the value of the token parameter in the Update message the Success message is in response to.

4.6. Error Message

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

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 => [ + suite ],
     ? supported-freshness-mechanisms => [ + freshness-mechanism ],
     ? versions => [ + version ],
     ? suit-reports => [ + suit-report ],
     * $$teep-error-extensions,
     * $$teep-option-extensions
  },
  err-code: uint (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 Update 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 ciphersuite(s) supported by the TEEP Agent. Details about the ciphersuite encoding can be found in Section 7. This field is optional but MUST be returned with the ERR_UNSUPPORTED_CRYPTO_ALG error message.

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 8. If this parameter is absent, it means only the nonce mechanism is supported.

versions
The versions parameter enumerates the TEEP protocol version(s) supported by the TEEP Agent. This otherwise optional parameter MUST be returned with the ERR_UNSUPPORTED_MSG_VERSION error message.

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

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 when there is no attestation evidence indicating that the error is genuine. In contrast, ERR_TEMPORARY_ERROR is an indication that a more agressive 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_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_CRYPTO_ALG (5)
The TEEP Agent does not support the cryptographic algorithm indicated in the request message. A TAM receiving this error might retry the request using a different cryptographic algorithm.

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. 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
version	3
ocsp-data	4
selected-cipher-suite	5
selected-version	6
evidence	7
tc-list	8
ext-list	9
manifest-list	10
msg	11
err-msg	12
evidence-format	13
requested-tc-list	14
unneeded-tc-list	15
component-id	16
tc-manifest-sequence-number	17
have-binary	18
suit-reports	19
token	20
supported-freshness-mechanisms	21

6. Behavior Specification

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

6.1. 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.

If a QueryResponse message is received that contains evidence, the evidence is passed to an attestation Verifier (see [I-D.ietf-rats-architecture]) to determine whether the Agent is in a trustworthy state. Based on the results of attestation, 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. 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.

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.

6.2. 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.

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 drops the message if it is not valid. 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 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.

7. Ciphersuites

A ciphersuite consists of an AEAD algorithm, a MAC algorithm, and a signature algorithm. Each ciphersuite is identified with an integer value, which corresponds to an IANA registered ciphersuite (see Section 10.2. This document specifies two ciphersuites.

Value	Ciphersuite
1	AES-CCM-16-64-128, HMAC 256/256, X25519, EdDSA
2	AES-CCM-16-64-128, HMAC 256/256, P-256, ES256

A TAM MUST support both ciphersuites. A TEEP Agent MUST support at least one of the two but can choose which one. For example, a TEEP Agent might choose ciphersuite 2 if it has hardware support for it.

Any ciphersuites 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].

8. Freshness Mechanisms

A freshness mechanism determines how a TAM can tell whether evidence provided in a Query Response 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 Section 10.3. This document defines the following freshness mechanisms:

Value	Freshness mechanism
1	Nonce
2	Timestamp
3	Epoch ID

In the Nonce mechanism, the evidence MUST include a nonce provided in the QueryRequest challenge. In other mechanisms, a timestamp or epoch ID determined via mechanisms outside the TEEP protocol is used, and the challenge is only needed in the QueryRequest message if a challenge is needed in generating evidence for reasons other than freshness.

9. 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 can rely on the attestation evidence provided by the TEEP Agent. To sign the attestation evidence, it is necessary for the device to possess a public key (usually in the form of a certificate [RFC5280]) along with the corresponding private key. Depending on the properties of the attestation mechanism, it is possible to uniquely identify a device based on information in the attestation evidence or in the certificate used to sign the attestation evidence. This uniqueness may raise privacy concerns. To lower the privacy implications the TEEP Agent MUST present its attestation evidence only to an authenticated and authorized TAM and when using EATS, 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 EATs is used, it is up to that mechanism to specify how privacy is provided.

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 symmetric encryption.

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 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

The QueryRequest message from a TAM to the TEEP Agent can include OCSP stapling data for the TAM's certificate and for intermediate CA certificates up to, but not including, the trust anchor so that the TEEP Agent can verify the certificate's revocation status. A certificate revocation status check on a Trusted Component Signer certificate is OPTIONAL by a TEEP Agent. A TAM is responsible for vetting a Trusted Component and before distributing them to TEEP Agents, so TEEP Agents can instead simply trust that a Trusted Component Signer certificate's status was done by the TAM.

CA Compromise
The CA issuing certificates to a TAM or a Trusted Component Signer might get compromised. A compromised intermediate CA certificate can be detected by a TEEP Agent by using OCSP information, assuming the revocation information is available. Additionally, it is RECOMMENDED to provide a way to update the trust anchor store used by the TEE, for example using a firmware update mechanism. 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.

Compromised TAM
The TEEP Agent SHOULD use OCSP information to verify the validity of the TAM's certificate (as well as the validity of intermediate CA certificates). The integrity and the accuracy of the clock within the TEE determines the ability to determine an expired or revoked certificate. OCSP stapling data includes signature generation time, allowing certificate validity dates to be compared to the current time.

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.

10. IANA Considerations

10.1. 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

10.2. Ciphersuite Registry

IANA is also requested to create a new registry for ciphersuites, as defined in Section 7.

10.3. Freshness Mechanism Registry

IANA is also requested to create a new registry for freshness mechanisms, as defined in Section 8.

10.4. CBOR Tag Registry

IANA is requested to register a CBOR tag in the "CBOR Tags" registry for use with TEEP messages.

The registry contents is:

- o CBOR Tag: TBD1
- o Data Item: TEEP Message
- o Semantics: TEEP Message, as defined in draft-ietf-teep-protocol (TODO: replace with RFC once published)
- o Reference: draft-ietf-teep-protocol (TODO: replace with RFC once published)
- o Point of Contact: TEEP working group (teep@ietf.org)

11. References

11.1. Normative References

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-12 (work in progress), April 2021.

[I-D.ietf-rats-eat]
Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", draft-ietf-rats-eat-10 (work in progress), June 2021.

[I-D.ietf-suit-manifest]
Moran, B., Tschofenig, H., Birkholz, H., and K. Zandberg, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", draft-ietf-suit-manifest-14 (work in progress), July 2021.

[I-D.moran-suit-report]
Moran, B., "Secure Reporting of Update Status", draft-moran-suit-report-01 (work in progress), February 2021.

[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>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <https://www.rfc-editor.org/info/rfc6960>.

[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>.

11.2. Informative References

[I-D.birkholz-rats-suit-claims]
Birkholz, H. and B. Moran, "Trustworthiness Vectors for the Software Updates of Internet of Things (SUIT) Workflow Model", draft-birkholz-rats-suit-claims-02 (work in progress), July 2021.

[I-D.ietf-teep-architecture]
Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", draft-ietf-teep-architecture-15 (work in progress), July 2021.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

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

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

SUIT_Envelope = any

teep-message-framework = [
  type: uint (0..23) / $teep-type-extension,
  options: { * teep-option },
  * uint; further integers, 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-query-request = 1
TEEP-TYPE-query-response = 2
TEEP-TYPE-update = 3
TEEP-TYPE-teep-success = 5
TEEP-TYPE-teep-error = 6

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

; data items as bitmaps
data-item-requested = $data-item-requested .within uint .size 8
attestation = 1
$data-item-requested /= attestation
trusted-components = 2
$data-item-requested /= trusted-components
extensions = 4
$data-item-requested /= extensions
suit-commands = 8
$data-item-requested /= suit-commands

query-request = [
  type: TEEP-TYPE-query-request,
  options: {
    ? token => bstr .size (8..64),
    ? supported-cipher-suites => [ + suite ],
    ? supported-freshness-mechanisms => [ + freshness-mechanism ],
    ? challenge => bstr .size (8..512),
    ? versions => [ + version ],
```

```
       ? ocsp-data => bstr,
       * $$query-request-extensions
       * $$teep-option-extensions
     },
     data-item-requested: data-item-requested
   ]

   ; ciphersuites
   suite = $TEEP-suite .within uint .size 4

   TEEP-AES-CCM-16-64-128-HMAC256--256-X25519-EdDSA = 1
   TEEP-AES-CCM-16-64-128-HMAC256--256-P-256-ES256  = 2

   $TEEP-suite /= TEEP-AES-CCM-16-64-128-HMAC256--256-X25519-EdDSA
   $TEEP-suite /= TEEP-AES-CCM-16-64-128-HMAC256--256-P-256-ES256

   ; freshness-mechanisms

   freshness-mechanism = $TEEP-freshness-mechanism .within uint .size 4

   FRESHNESS_NONCE = 0
   FRESHNESS_TIMESTAMP = 1
   FRESHNESS_EPOCH_ID = 2

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

   query-response = [
     type: TEEP-TYPE-query-response,
     options: {
       ? token => bstr .size (8..64),
       ? selected-cipher-suite => suite,
       ? selected-version => version,
       ? evidence-format => text,
       ? evidence => bstr,
       ? tc-list => [ + tc-info ],
       ? requested-tc-list => [ + requested-tc-info ],
       ? unneeded-tc-list => [ + SUIT_Component_Identifier ],
       ? ext-list => [ + ext-info ],
       * $$query-response-extensions,
       * $$teep-option-extensions
     }
   ]

   tc-info = {
     component-id => SUIT_Component_Identifier,
     ? tc-manifest-sequence-number => .within uint .size 8
```

```
}

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

update = [
  type: TEEP-TYPE-update,
  options: {
    ? token => bstr .size (8..64),
    ? 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 => [ + suite ],
     ? supported-freshness-mechanisms => [ + freshness-mechanism ],
     ? versions => [ + version ],
     ? suit-reports => [ + suit-report ],
     * $$teep-error-extensions,
     * $$teep-option-extensions
  },
  err-code: uint (0..23)
]

; The err-code parameter, uint (0..23)
ERR_PERMANENT_ERROR = 1
ERR_UNSUPPORTED_EXTENSION = 2
ERR_UNSUPPORTED_MSG_VERSION = 4
```

```
ERR_UNSUPPORTED_CRYPTO_ALG = 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
ocsp-data = 4
selected-cipher-suite = 5
selected-version = 6
evidence = 7
tc-list = 8
ext-list = 9
manifest-list = 10
msg = 11
err-msg = 12
evidence-format = 13
requested-tc-list = 14
unneeded-tc-list = 15
component-id = 16
tc-manifest-sequence-number = 17
have-binary = 18
suit-reports = 19
token = 20
```

D. Examples of Diagnostic Notation and Binary Representation

D.1. Some assumptions in examples

o OCSP stapling data = h'010203'

o TEEP Device will have two TCs with the following SUIT Component Identifiers:

  * [0x000102030405060708090a0b0c0d0e0f]

  * [0x100102030405060708090a0b0c0d0e0f]

o SUIT manifest-list is set empty only for example purposes

D.2. QueryRequest Message

D.2.1. CBOR Diagnostic Notation

```
/ query-request = /
[
  1,  / type : TEEP-TYPE-query-request = 1 (uint (0..23)) /
  / options : /
  {
    20 : 0xa0a1a2a3a4a5a6a7a8a9aaabacadaeaf,
             / token = 20 (mapkey) :
               h'a0a1a2a3a4a5a6a7a8a9aaabacadaeaf' (bstr .size (8..64)),
               generated by TAM /
    1 : [ 1 ], / supported-cipher-suites = 1 (mapkey) :
                  TEEP-AES-CCM-16-64-128-HMAC256--256-X25519-EdDSA =
                  [ 1 ] (array of .within uint .size 4) /
    3 : [ 0 ], / version = 3 (mapkey) :
                  [ 0 ] (array of .within uint .size 4) /
    4 : h'010203' / ocsp-data = 4 (mapkey) : 0x010203 (bstr) /
  },
  3   / data-item-requested :
        attestation | trusted-components = 3 (.within uint .size 8) /
]
```

D.2.2. CBOR Binary Representation

```
83                        # array(3)
  01                      # unsigned(1) uint (0..23)
  A4                      # map(4)
    14                    # unsigned(20) uint (0..23)
    4F                    # bytes(16) (8..64)
      A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
    01                    # unsigned(1) uint (0..23)
    81                    # array(1)
      01                  # unsigned(1) within uint .size 4
    03                    # unsigned(3) uint (0..23)
    81                    # array(1)
      00                  # unsigned(0) within uint .size 4
    04                    # unsigned(4) uint (0..23)
    43                    # bytes(3)
      010203              # "\x01\x02\x03"
  03                      # unsigned(3) .within uint .size 8
```

D.3. Entity Attestation Token

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

D.3.1. 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 /       <TBD>: 3, / secure-restricted /
    / device-identifier /    <TBD>: h'e99600dd921649798b013e9752dcf0c5',
    / vendor-identifier /    <TBD>: h'2b03879b33434a7ca682b8af84c19fd4',
    / class-identifier /     <TBD>: h'9714a5796bd245a3a4ab4f977cb8487f',
    / chip-version-scheme /  <TBD>: "MyTEE v1.0",
    / component-identifier / <TBD>: h'60822887d35e43d5b603d18bcaa3f08d',
    / version /              <TBD>: "v0.1"
}
```

D.4. QueryResponse Message

D.4.1. CBOR Diagnostic Notation

```
/ query-response = /
[
  2,  / type : TEEP-TYPE-query-response = 2 (uint (0..23)) /
  / options : /
  {
    20 : 0xa0a1a2a3a4a5a6a7a8a9aaabacadaeaf,
            / token = 20 (mapkey) :
              h'a0a1a2a3a4a5a6a7a8a9aaabacadaeaf' (bstr .size (8..64)),
              given from TAM's QueryRequest message /
    5 : 1,  / selected-cipher-suite = 5 (mapkey) :
              TEEP-AES-CCM-16-64-128-HMAC256--256-X25519-EdDSA =
              1 (.within uint .size 4) /
    6 : 0,  / selected-version = 6 (mapkey) :
              0 (.within uint .size 4) /
    7 : ... / evidence = 7 (mapkey) :
              Entity Attestation Token /
    8 : [   / tc-list = 8 (mapkey) : (array of tc-info) /
      {
        16 : [ 0x000102030405060708090a0b0c0d0e0f ] / component-id =
               16 (mapkey) : [ h'000102030405060708090a0b0c0d0e0f' ]
               (SUIT_Component_Identifier =  [* bstr]) /
      },
      {
        16 : [ 0x100102030405060708090a0b0c0d0e0f ] / component-id =
               16 (mapkey) : [ h'100102030405060708090a0b0c0d0e0f' ]
               (SUIT_Component_Identifier =  [* bstr]) /
      }
        ]
    }
]
```

D.4.2. CBOR Binary Representation

```
82                        # array(2)
  02                      # unsigned(2) uint (0..23)
  A5                      # map(5)
    14                    # unsigned(20) uint (0..23)
    4F                    # bytes(16) (8..64)
      A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
    05                    # unsigned(5) uint (0..23)
    01                    # unsigned(1) .within uint .size 4
    06                    # unsigned(6) uint (0..23)
    00                    # unsigned(0) .within uint .size 4
    07                    # unsigned(7) uint (0..23)
      ...                 # Entity Attestation Token
    08                    # unsigned(8) uint (0..23)
    82                    # array(2)
      81                  # array(1)
        4F                # bytes(16)
          000102030405060708090A0B0C0D0E0F
      81                  # array(1)
        4F                # bytes(16)
          100102030405060708090A0B0C0D0E0F
```

D.5. Update Message

D.5.1. CBOR Diagnostic Notation

```
/ update = /
[
  3,  / type : TEEP-TYPE-update = 3 (uint (0..23)) /
  / options : /
  {
    20 : 0xa0a1a2a3a4a5a6a7a8a9aaabacadaeaf,
             / token = 20 (mapkey) :
               h'a0a1a2a3a4a5a6a7a8a9aaabacadaeaf' (bstr .size (8..64)),
               generated by TAM /
    10 : [ ] / manifest-list = 10 (mapkey) :
               [ ] (array of bstr wrapped SUIT_Envelope(any)) /
             / empty, example purpose only /
  }
]
```

D.5.2. CBOR Binary Representation

```
   82                        # array(2)
     03                      # unsigned(3) uint (0..23)
     A3                      # map(3)
       14                    # unsigned(20) uint (0..23)
       4F                    # bytes(16) (8..64)
         A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
       0A                    # unsigned(10) uint (0..23)
       80                    # array(0)
```

D.6. Success Message

D.6.1. CBOR Diagnostic Notation

```
/ teep-success = /
[
  5,  / type : TEEP-TYPE-teep-success = 5 (uint (0..23)) /
  / options : /
  {
    20 : 0xa0a1a2a3a4a5a6a7a8a9aaabacadaeaf,
             / token = 20 (mapkey) :
               h'a0a1a2a3a4a5a6a7a8a9aaabacadaeaf' (bstr .size (8..64)),
               given from TAM's Update message /
  }
]
```

D.6.2. CBOR Binary Representation

```
   82                        # array(2)
     05                      # unsigned(5) uint (0..23)
     A1                      # map(1)
       14                    # unsigned(20) uint (0..23)
       4F                    # bytes(16) (8..64)
         A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
```

D.7. Error Message

D.7.1. CBOR Diagnostic Notation

```
 / teep-error = /
 [
   6,  / type : TEEP-TYPE-teep-error = 6 (uint (0..23)) /
   / options : /
   {
     20 : 0xa0a1a2a3a4a5a6a7a8a9aaabacadaeaf,
            / token = 20 (mapkey) :
              h'a0a1a2a3a4a5a6a7a8a9aaabacadaeaf' (bstr .size (8..64)),
              given from TAM's Update message /
     12 : "disk-full"  / err-msg = 12 (mapkey) :
                         "disk-full" (text .size (1..128)) /
   },
   17, / err-code : ERR_MANIFEST_PROCESSING_FAILED = 17 (uint (0..23)) /
 ]
```

D.7.2. CBOR binary Representation

```
   83                       # array(3)
     06                     # unsigned(6) uint (0..23)
     A2                     # map(2)
       14                   # unsigned(20) uint (0..23)
       4F                   # bytes(16) (8..64)
         A0A1A2A3A4A5A6A7A8A9AAABACADAEAF
       0C                   # unsigned(12) uint (0..23)
       69                   # text(9) (1..128)
         6469736B2D66756C6C # "disk-full"
     11                     # unsigned(17) uint (0..23)
```

Authors' Addresses

Hannes Tschofenig
Arm Ltd.
Absam, Tirol 6067
Austria

Email: hannes.tschofenig@arm.com

Mingliang Pei
Broadcom
350 Ellis St
Mountain View, CA 94043
USA

Email: mingliang.pei@broadcom.com

David Wheeler
Amazon
US

Email: davewhee@amazon.com

Dave Thaler
Microsoft
US

Email: dthaler@microsoft.com

Akira Tsukamoto
AIST
JP

Email: akira.tsukamoto@aist.go.jp