Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: January 18, 2018

M. Pei
Symantec
N. Cook
Intercede
M. Yoo
Solacia
A. Atyeo
Intercede
H. Tschofenig
ARM Ltd.
July 17, 2017

The Open Trust Protocol (OTrP)
draft-pei-opentrustprotocol-04.txt

Abstract

This document specifies the Open Trust Protocol (OTrP), a protocol to install, update, and delete applications in a Trusted Execution Environment (TEE) and to manage their security configuration ~~in a Trusted Execution Environment (TEE)~~.

TEEs are used in environments where security services should be isolated from a regular operating system (often called a rich OS). This form of compartmentalization grants a smaller codebase access to security sensitive services and restricts communication from the rich OS to those security services via mediated access.

Commented [DT1]: Suggested rewording

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 5
2. Requirements Language 6
3. Terminology 6
  3.1. Definitions 6
  3.2. Abbreviations 7
4. OTrP Entities and Trust Model 8
  4.1. System Components 8
  4.2. Trusted Anchors in TEE 9
  4.3. Trusted Anchors in TSM 9
  4.4. Keys and Cerificate Types 9
5. Protocol Scope and Entity Relations 12
  5.1. A Sample Device Setup Flow 14
  5.2. Derived Keys in the Protocol 14
  5.3. Security Domain Hierarchy and Ownership 15
  5.4. SD Owner Identification and TSM Certificate Requirements 16
  5.5. Service Provider Container 16
6. OTrP Agent 17
  6.1. Role of OTrP Agent 17
  6.2. OTrP Agent and Global Platform TEE Client API 18
  6.3. OTrP Agent Implementation Consideration 18
    6.3.1. OTrP Agent Distribution 18
    6.3.2. Number of OTrP Agent 18
    6.3.3. OTrP Android Service Option 19
  6.4. OTrP Agent API for Client Applications 19
    6.4.1. API processMessage 19
    6.4.2. API getTAInformation 20
  6.5. Sample End-to-End Client Application Flow 22
    6.5.1. Case 1: A new Client App uses a TA 22
    6.5.2. Case 2: A previously installed Client Application calls a TA 24

7. OTrP Messages 25
7.1. Message Format 25
7.2. Message Naming Convention 25
7.3. Request and Response Message Template 26
7.4. Signed Request and Response Message Structure 26
7.4.1. Identifying signing and Encryption keys for JWS/JWE messaging 28
7.5. JSON Signing and Encryption Algorithms 28
7.5.1. Supported JSON Signing Algorithms 30
7.5.2. Support JSON Encryption Algorithms 30
7.5.3. Supported JSON Key Management Algorithms 30
7.6. Common Errors 31
7.7. OTrP Message List 31
7.8. OTrP Request Message Routing Rules 32
7.8.1. SP Anonymous Attestation Key (SP AIK) 32
8. Detailed Messages Specification 32
8.1. GetDeviceState 33
8.1.1. GetDeviceStateRequest message 33
8.1.2. Request processing requirements at a TEE 34
8.1.3. Firmware signed data 35
8.1.3.1. Supported Firmware Signature Methods 35
8.1.4. Post Conditions 36
8.1.5. GetDeviceStateResponse message 36
8.1.6. Error Conditions 40
8.1.7. TSM Processing Requirements 41
8.2. Security Domain Management 42
8.2.1. CreateSD 42
8.2.1.1. CreateSDRequest Message 42
8.2.1.2. Request processing requirements at a TEE 45
8.2.1.3. CreateSDResponse Message 46
8.2.1.4. Error Conditions 47
8.2.2. UpdateSD 48
8.2.2.1. UpdateSDRequest Message 48
8.2.2.2. Request processing requirements at a TEE 51
8.2.2.3. UpdateSDResponse Message 53
8.2.2.4. Error Conditions 54
8.2.3. DeleteSD 55
8.2.3.1. DeleteSDRequest Message 55
8.2.3.2. Request processing requirements at a TEE 57
8.2.3.3. DeleteSDResponse Message 58
8.2.3.4. Error Conditions 60
8.3. Trusted Application Management 60
8.3.1. InstallTA 60
8.3.1.1. InstallTARequest Message 62
8.3.1.2. InstallTAResponse Message 64
8.3.1.3. Error Conditions 65
8.3.2. UpdateTA 65
8.3.2.1. UpdateTARequest Message 67

8.3.2.2. UpdateTAResponse Message 68
8.3.2.3. Error Conditions 70
8.3.3. DeleteTA 70
8.3.3.1. DeleteTARequest Message 70
8.3.3.2. Request processing requirements at a TEE 72
8.3.3.3. DeleteTAResponse Message 73
8.3.3.4. Error Conditions 74
9. Response Messages a TSM May Expect 74
10. Basic Protocol Profile 75
11. Attestation Implementation Consideration 76
11.1. OTrP Secure Boot Module 76
11.1.1. Attestation signer 76
11.1.2. SBM initial requirements 76
11.2. TEE Loading 77
11.3. Attestation Hierarchy 77
11.3.1. Attestation hierarchy establishment: manufacture . . 78
11.3.2. Attestation hierarchy establishment: device boot . . 78
11.3.3. Attestation hierarchy establishment: TSM 78
12. Acknowledgements 78
13. Contributors 79
14. IANA Considerations 79
14.1. Error Code List 79
15. Security Consideration 80
15.1. Cryptographic Strength 81
15.2. Message Security 81
15.3. TEE Attestation 81
15.4. TA Protection 82
15.5. TA Personalization Data 82
15.6. TA trust check at TEE 82
15.7. One TA Multiple SP Case 83
15.8. OTrP Agent Trust Model 83
15.9. OCSP Stapling Data for TSM signed messages 83
15.10. Data protection at TSM and TEE 84
15.11. Privacy consideration 84
15.12. Threat mitigation 84
15.13. Compromised CA 85
15.14. Compromised TSM 85
15.15. Certificate renewal 85
16. References 85
16.1. Normative References 85
16.2. Informative References 86
Appendix A. Sample Messages 86
A.1. Sample Security Domain Management Messages 86
A.1.1. Sample GetDeviceState 86
A.1.1.1. Sample GetDeviceStateRequest 86
A.1.1.2. Sample GetDeviceStateResponse 87
A.1.2. Sample CreateSD 90
A.1.2.1. Sample CreateSDRequest 90

A.1.2.2. Sample CreateSDResponse 93
A.1.3. Sample UpdateSD 94
A.1.3.1. Sample UpdateSDRequest 95
A.1.3.2. Sample UpdateSDResponse 96
A.1.4. Sample DeleteSD 96
A.1.4.1. Sample DeleteSDRequest 96
A.1.4.2. Sample DeleteSDResponse 98
A.2. Sample TA Management Messages 100
A.2.1. Sample InstallTA 100
A.2.1.1. Sample InstallTARequest 100
A.2.1.2. Sample InstallTAResponse 101
A.2.2. Sample UpdateTA 103
A.2.2.1. Sample UpdateTARequest 103
A.2.2.2. Sample UpdateTAResponse 104
A.2.3. Sample DeleteTA 107
A.2.3.1. Sample DeleteTARequest 107
A.2.3.2. Sample DeleteTAResponse 109
Authors' Addresses 111

1. Introduction

The Trusted Execution Environment (TEE) concept has been designed and used to increase security by separating a regular operating system~~s~~, also referred to as a Rich Execution Environment (REE), from security-sensitive applications. In an TEE ecosystem, a Trust Service Manager (TSM) is used to ~~authorize~~ manage keys and the Trusted Applications (TA) that run in a device. Different device vendors may use different TEE implementations. Different application providers may use different TSM providers. There arises a need of an open interoperable protocol that allows a trustworthy TSM to manage Security Domains and contents running in different ~~Trusted Execution Environment (TEE)~~TEEs of various devices.

The Open Trust Protocol (OTrP) defines a protocol between a TSM and a TEE and relies on IETF-defined end-to-end security mechanisms, namely JSON Web Encryption (JWE), JSON Web Signature (JWS), and JSON Web Key (JWK).

This specification assumes that a device that utilizes this specification is equipped with a TEE and is pre-provisioned with a device-unique public/private key pair, which is securely stored. This key pair is referred as the 'root of trust'. A Service Provider (SP) uses such a device to run Trusted Applications (TA).

A security domain is defined as the TEE representation of a service provider and is a logical space that contains the service provider's Trusted Applications. Each security domain requires the management

Commented [DT2]: Question: so the 'root of trust' is the device's own key pair? If not, then the device would not have the private key as this sentence claims.

Commented [DT3]: This term is not defined until later, so needs to be clarified here. E.g.
"An entity that uses such a device to run Trusted Applications (TAs) is known as a Service Provider."

Commented [DT4]: Some places in the doc capitalize Service Provider. Some places don't. Be consistent.

operations of Trusted Applications (TAs) in the form of installation, update and deletion.

The protocol builds on the following properties of the system:

1. The SP needs to determine security-relevant information of a device before provisioning information to a TEE. Examples include the verification of the root of trust, the type of firmware installed, and the type of TEE included in a device.

2. A TEE in a device needs to determine whether a SP or the TSM is authorized to manage applications in the TEE.

3. Secure Boot must be able to ensure a TEE is genuine.

This specification defines message payloads exchanged between devices and a TSM but does not mandate a specific transport.

Commented [DT5]: We have to provide at least one that's MTI or we can't get interoperability.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Terminology

3.1. Definitions

The definitions provided below are defined as used in this document. The same terms may be defined differently in other documents.

Client Application: An application running on a rich OS, such as an Android, Windows, or iOS application, provided by a SP.

Commented [DT6]: Do we need to allow a scenario where a client application is provided by an entity other than the SP?

Device: A physical piece of hardware that hosts symmetric key cryptographic modules

Commented [DT7]: Why not just say "a TEE"?

OTrP Agent: An application running in the rich OS allowing communication with the TSM and the TEE.

Rich Application: Alternative name of "Client Application". In this document we may use these two terms interchangably.

Commented [DT8]: Why do we need two terms? Can't we just pick one?

Rich Execution Environment (REE) An environment that is provided and governed by a rich OS, potentially in conjunction with other supporting operating systems and hypervisors; it is outside of the TEE. This environment and applications running on it are considered un-trusted.

Secure Boot Module (SBM): A firmware in a device that delivers secure boot functionality. It is also referred as Trusted Firmware (TFW) in this document.

Service Provider (SP): An entity that wishes to supply Trusted Applications to remote devices. A Service Provider requires the help of a TSM in order to provision the Trusted Applications to the devices.

Trust Anchor: A root certificate that a module trusts. It is usually embedded in one validating module, and used to validate the trust of a remote entity's certificate.

Trusted Application (TA): An application that runs in a TEE.

Trusted Execution Environment (TEE): An execution environment that runs alongside of, but is isolated from, an REE. A TEE has security capabilities and meets certain security-related requirements: It protects TEE assets from general software attacks, defines rigid safeguards as to data and functions that a program can access, and resists a set of defined threats. There are multiple technologies that can be used to implement a TEE, and the level of security achieved varies accordingly.

3.2. Abbreviations

CA Certificate Authority

OTrP Open Trust Protocol

Commented [DT9]: Undefined term. And I think the real definition should be the part after the ";"
The rest is just an example implementation

Commented [DT10]: Just pick one term and use it everywhere, except in the glossary where it's fine to say the term is synonymous with some other term that might be used in other documents or literature in the industry.

Commented [DT11]: Undefined term

Commented [DT12]: Personally I think a TEE should be explicitly defined as having at least the following three properties:

1)A unique security identity that cannot be cloned
2)Assurance that only authorized code can run in the TEE.
3)Memory that cannot be read by code outside the TEE

REE Rich Execution Environment

SD Security Domain

SP Service Provider

SBM Secure Boot Module

TA Trusted Application

TEE Trusted Execution Environment

TFW Trusted Firmware

TSM Trusted Service Manager

4. OTrP Entities and Trust Model

4.1. System Components

~~There are t~~The following are the main components in this OTrP system.

TSM: The TSM is responsible for originating and coordinating lifecycle management activity on a particular TEE.

A Trust Service Manager (TSM) ~~is at the core to the protocol that~~ manages device trust check on behalf of service providers~~ for the ecosystem scalability~~. In addition to its device trust management for a service provider, the TSM provides Security Domain management and TA management in a device, in particular~~ly~~, over-the-air update to keep Trusted Applications up to date and clean up when a version should be removed.

In the context of this specification, the term Trusted Application Manager (TAM) and TSM are synonymous.

Certificate Authority (CA): Mutual trust between a device and a TSM as well as a Service Provider is based on certificates. A device embeds a list of root certificates, called Trust Anchors, from trusted Certificate Authorities that a TSM will be validated against. A TSM will remotely attest a device by checking whether a device comes with a certificate from a trusted CA.

TEE: The TEE resides in the device chip security zone and is responsible for protecting applications from attack, enabling the application to perform secure operations.

Commented [DT13]: I can't parse this

Commented [DT14]: Then pick one and only use one of them.

Commented [DT15]: Undefined term. Why is this phrase needed at all given there's a definition in the glossary?

REE: The REE, ~~usually called device OS such as Android OS in a phone device,~~ is responsible for enabling off device communications to be established between the TEE and TSM. OTrP does not require the device OS to be secure.

OTrP Agent: An application in the REE that can relay messages between a Client Application and TEE.

Secure Boot: Secure boot (for the purposes of OTrP) must enable authenticity checking of TEEs by the TSM.

The OTrP establishes appropriate trust anchors to enable TEE and TSMs to communicate in a trusted way when performing lifecycle management transactions. The main trust relationships between the components are the following.

1. TSM must be able to ensure a TEE is genuine

2. TEE must be able to ensure a TSM is genuine

3. Secure Boot must be able to ensure a TEE is genuine

4.2. Trusted Anchors in TEE

The TEE in each device comes with a trust store that contains a whitelist of the TSM's root CA certificates, which are called Trust Anchors. A TSM will be trusted to manage Security Domains and TAs in a device only if its certificate is chained to one of the root CA certificates in this trust store.

Such a list is typically embedded in the TEE of a device, and the list update is enabled and handled by the device OEM ~~provider~~.

4.3. Trusted Anchors in TSM

The Trust Anchor set in a TSM consists of a list of Certificate Authority certificates that signs various device TEE certificates. A TSM decides what TEE and TFW it will trust.

4.4. Keys and Certificate Types

OTrP ~~Protocol~~ leverages the following list of trust anchors and identities in generating signed and encrypted command messages that are exchanged between a device's ~~with~~ TEE and a TSM. With these security artifacts, OTrP Messages are able to deliver end-to-end security without relying on any transport security.

Commented [DT16]: Don't need to repeat information already in the glossary. In fact, I'd recommend not having the same terms in both places. Also don't call out any single OS or vendor, even in an example, unless it's unique in the respect mentioned. Either mention multiple or none.

Commented [DT17]: The OTrP Agent? Or just "OTrP"?

Commented [DT18]: Redundant with end of section 1. Remove duplication.

Commented [DT19]: Use consistent capitzliation throughout. Some places it's "security domains" and other places "Security Domains"

Commented [DT20]: What is "it" referring to? The TSM? One of the TAs?

Commented [DT21]: This sounds way too phone specific, and I disagree with this in general. I believe it should be handled by either the device *admin* or the TEE silicon vendor, or it's not secure (or at least much less secure).

Key Entity Name	Location	Issuer	Trust Implication	Cardinality
1. TFW keypair and Certificate	Device secure storage	OEM CA	A white list of FW root CA trusted by TSMs	1 per device
2. TEE keypair and Certificate	Device TEE	TEE CA under a root CA	A white list of TEE root CA trusted by TSMs	1 per device
3. TSM keypair and Certificate	TSM provider	TSM CA under a root CA	A white list of TSM root CA embedded in TEE	1 or multiple can be used by a TSM
4. SP keypair and Certificate	SP	SP signer CA	TSM manages SP. TA trust is delegated to TSM. TEE trusts TSM to ensure that a TA is trustworthy.	1 or multiple can be used by a TSM

Table 1: Key and Certificate Types

Commented [DT22]: Again this looks suspect.

Commented [DT23]: Two words

1. TFW keypair and Certificate: A key pair and certificate for evidence of secure boot and trustworthy firmware in a device.

   Location: Device secure storage

   Supported Key Type: RSA and ECC

   Issuer: OEM CA

   Trust Implication: A white list of FW root CA trusted by TSMs

   Cardinality: One per device

2. TEE keypair and Certificate: It is used for device attestation to remote TSM and SP.

This key pair is burned into the device at device manufacturer. The key pair and its certificate are valid for the expected lifetime of the device.

Location: Device TEE

Supported Key Type: RSA and ECC

Issuer: TEE CA that chains to a root CA

Trust Implication: A white list of TEE root CA trusted by TSMs

Cardinality: One per device

3. TSM keypair and Certificate: A TSM provider acquires a certificate from a CA that a TEE trusts.

Location: TSM provider

Supported Key Type: RSA and ECC.

Supported Key Size: RSA 2048-bit, ECC P-256 and P-384.

Issuer: TSM CA that chains to a root CA

Trust Implication: A white list of TSM root CA embedded in TEE

Cardinality: One or multiple can be used by a TSM

4. SP keypair and Certificate: A SP uses its own key pair and certificate to sign a TA.

Location: SP

Supported Key Type: RSA and ECC

Supported Key Size: RSA 2048-bit, ECC P-256 and P-384

Issuer: SP signer CA that chains to a root CA

Trust Implication: TSM manages SP. TA trust is delegated to TSM. TEE trusts the TSM to ensure that a TA is trustworthy.

Commented [DT24]: Is this referring to the roots configured in the TEE or in t TSM? (which can be different)

Commented [DT25]: Need to make sure the protocol supports crypto agility. See RFC 6421.

Commented [DT26]: What does this mean? Delegated by whom?

Cardinality: One or multiple can be used by a SP

5. Protocol Scope and Entity Relations

This document specifies the minimally required interoperable artifacts to establish mutual trust between a TEE and a TSM. The protocol provides specifications for the following three entities:

Commented [DT27]: Meaning what?

1. Key and certificate types required for device firmware, TEEs, TAs, SPs, and TSMs

2. Data message formats that should be exchanged between a TEE in a device and a TSM

3. An OTrP Agent application in the REE that can relay messages between a Client Application and a TEE

Figure 1: Protocol Scope and Entity Relationship

```
   PKI    CA    --CA                                   CA--
           |    |                                         |
           |    |                                         |
           |    |                                         |
   Device  |    |   ----OTrP Agent --- Rich App ---       |
   SW      |    |   |                             |       |
           |    |   |                             |       |
           |    |   |                             |       |
   OTrP    |    -- TEE                        TSM-------
           |
           |
          FW
```

Figure 2: OTrP System Diagram

```
                ---OTrP Message Protocol--
                |                        |
                |                        |
    --------------------          ---------------    ----------
    |  REE   |  TEE    |          |    TSM      |    |  SP    |
    |  ---   |  ---    |          |    ---      |    |  --    |
    |        |         |          |             |    |        |
    | Client | SD (TAs)|          |   SD / TA   |    |  TA    |
    |  Apps  |         |          |    Mgmt     |    |        |
    |   |    |         |          |             |    |        |
    |   |    |         |          |             |    |        |
    | OTrP   | Trusted |          |  Trusted    |    |        |
    | Agent  |  CAs    |          | FW, TEE CAs |    |        |
    |        |         |          |             |    |        |
    |        |TEE Key/ |          |  TSM Key/   |    |SP Key/ |
    |        |  Cert   |          |    Cert     |    | Cert   |
    |        | FW Key/ |          |             |    |        |
    |        |  Cert   |          |             |    |        |
    ------------------            ---------------    ----------
                |                        |              |
                |                        |              |
                -----------------------------------------
                                  |
                                  |
                            --------------
                            |    CA      |
                            --------------
```

In the previous diagram, different Certificate Authorities can be used respectively for different types of certificates. OTrP Messages are always signed, where the signer keys is the message creator's key pair such as a FW key pair, TEE key pair or TSM key pair.

The main OTrP Protocol component is the set of standard JSON messages created by TSM to deliver device SD and TA management commands to a device, and device attestation and response messages created by a TEE to respond to TSM OTrP Messages.

The communication method of OTrP Messages between a TSM and TEE in a device is left to TSM providers for maximal interoperability. A TSM can work with its SP and Client Applications how it gets OTrP Messages from a TSM. When a Client Application has had an OTrP Message from its TSM, it is imperative to have an interoperable interface to communicate with various TEE types. This is the OTrP Agent interface that serves this purpose. The OTrP Agent doesn't need to know the actual content of OTrP Messages except for the TEE routing information.

Commented [DT28]: Can't parse this. Do you mean the message are signed by the message creator's *private* key?

Commented [DT29]: JSON can be pretty verbose to try to apply to a TEE in a constrained device. Can CBOR be used on the wire instead? (since there's a simple transform from JSON to CBOR already defined)

Commented [DT30]: Disagree, this results in *lack* of maximal interoperability. The TSM and the TEE may be from different vendors as the draft says earlier. So the transport protocol (the thing that carries the JSON messages inside it) between a TSM and OTrP agent must be standardized.

Commented [DT31]: Section 3.1 says a Client App is what runs in the REE not the TEE. How is it related to OTrP messages? The diagram above does not show the Client App being involved in OTrP messages at all (at most, the OTrP Agent is, not the Client App per se).

5.1. A Sample Device Setup Flow

Step 1: Prepare Images for Devices

1. [TEE vendor] Deliver TEE Image (CODE Binary)

2. [CA] Deliver root CA Whitelist

3. [Soc] Deliver TFW Image

Step 2: Inject Key Pairs and Images to Devices

1. [OEM] Generate Secure Boot Key Pair (May be shared among multiple devices)

2. [OEM] Flash signed TFW Image and signed TEE Image onto devices (signed by Secure Boot Key)

Step 3: Set up attestation key pair in devices

1. [OEM] Flash Secure Boot Public Key and eFuse Key (eFuse key is unique per device)

2. [TFW/TEE] Generate a unique attestation key pair and get a certificate for the device.

Step 4: Set up trust anchors in devices

1. [TFW/TEE] Store the key and certificate encrypted with the eFuse key

2. [TEE vendor or OEM] Store trusted CA certificate list into devices

5.2. Derived Keys in the Protocol

The protocol generates the following two key pairs in run time to assist message communication and anonymous verification between a TSM and TEE.

1. TEE Anonymous Key (TEE AIK): one derived key pair per TEE in a device

The purpose of the key pair is to sign data by a TEE without using its TEE device key for anonymous attestation to a Client Application. This key is generated in the first GetDeviceState query. The public key of the key pair is returned to the caller Client Application for future TEE returned data validation.

Commented [DT32]: To whom?

2. TEE SP AIK: one derived key per SP in a device

The purpose of this key pair is for a TSM to encrypt TA binary data when it sends a TA to a device for installation. This key is generated in the first SD creation for a SP. It is deleted when all SDs are removed for a SP in a device.

With the presence of a TEE SP AIK, it isn't necessary to have a shared SP independent TEE AIK. For the initial release, this specification will not use TEE AIK.

Commented [DT33]: Then remove it from this document.

5.3. Security Domain Hierarchy and Ownership

The primary job of a TSM is to help a SP to manage its trusted applications. A TA is typically installed in an SD. An SD is commonly created for a SP.

When an SP delegates its SD and TA management to a TSM, an SD is created on behalf of a TSM in a TEE and the owner of the SD is assigned to the TSM. An SD may be associated with a SP but the TSM has full privilege to manage the SD for the SP.

Each SD for an SP is associated with only one TSM. When an SP changes TSM, a new SP SD must be created to associate with the new TSM. The TEE will maintain a registry of TSM ID and SP SD ID mapping.

From an SD ownership perspective, the SD tree is flat and there is only one level. An SD is associated with its owner. It is up to the TEE~~'s~~ implementation how it maintains SD binding information for a TSM and different SPs under the same TSM.

Commented [DT34]: Why do you need this complexity? Just say there's one TSM per SP, and then you don't need two different IDs. If you do need to, then you need to motivate why the extra complexity is needed.

It is an important decision in this protocol specification that a TEE doesn't need to know whether a TSM is authorized to manage the SD for an SP. This authorization is implicitly triggered by an SP Client Application, which instructs what TSM it wants to use. An SD is always associated with a TSM in addition to its SP ID. A rogue TSM isn't able to do anything on an unauthorized SP's SD managed by another TSM.

Commented [DT35]: It would seem more natural to me to state what TA it wants to use, and infer the TSM from that.

Since a TSM may support multiple SPs, sharing the same SD name for different SPs creates a dependency in deleting a SD. A SD can be deleted only after all TAs associated with this SD is deleted. A SP cannot delete a Security Domain on its own with a TSM if a TSM decides to introduce such sharing. There are cases where multiple virtual SPs belong to the same organization, and a TSM chooses to use the same SD name for those SPs. This is totally up to the TSM implementation and out of scope of this specification.

Commented [DT36]: Why not just disallow this?

5.4. SD Owner Identification and TSM Certificate Requirements

There is a need of cryptographically binding proof about the owner of an SD in a device. When an SD is created on behalf of a TSM, a future request from the TSM must present itself as a way that the TEE can verify it is the true owner. The certificate itself cannot reliably used as the owner because TSM may change its certificate.

To this end, each TSM will be associated with a trusted identifier defined as an attribute in the TSM certificate. This field is kept the same when the TSM renew its certificates. A TSM CA is responsible to vet the requested TSM attribute value.

This identifier value must not collide among different TSM providers, and one TSM shouldn't be able to claim the identifier used by another TSM provider.

The certificate extension name to carry the identifier can initially use SubjectAltName:registeredID. A dedicated new extension name may be registered later.

One common choice of the identifier value is the TSM's service URL. A CA can verify the domain ownership of the URL with the TSM in the certificate enrollment process.

TEE can assign this certificate attribute value as the TSM owner ID for the SDs that are created for the TSM.

An alternative way to represent a SD ownership by a TSM is to have a unique secret key upon SD creation such that only the creator TSM is able to produce a Proof-of-Possession (POP) data with the secret.

5.5. Service Provider Container

A sample Security Domain hierarchy for the TEE is shown below.

```
      ----------
      |  TEE   |
      ----------
          |
          |          ---------------
          |----------| SP1 Root SD |
          |          ---------------
          |                 |
          |                 |          --------------
          |                 |----------| SP1 Sub SD |
          |                 |          --------------
          |                 |          --------------
          |                 |----------| SP1 Sub SD |
          |                            --------------
          |          ---------------
          |----------| SP2 Root SD |
                     ---------------
```

The OTrP assumes that a SP managed by TSM1 cannot be managed by TSM2. Explicit permission grant should happen. SP can authorize TSM.

6. OTrP Agent

OTrP Agent is an Rich Application or SDK that facilitates communication between a TSM and TEE. It also provides interfaces for TSM SDK or Client Applications to query and trigger TA installation that the application needs to use.

This interface for Client Applications may be commonly an Android service call. A Client Application interacts with a TSM, and turns around to pass messages received from TSM to OTrP Agent.

In all cases, a Client Application needs to be able to identify an OTrP Agent that it can use.

6.1. Role of OTrP Agent

OTrP Agent is responsible to communicate with TEE. It takes request messages from an application. The input data is mostly from a TSM that an application communicates. An application may also directly call OTrP Agent for some TA query functions.

OTrP Agent may internally process a request from TSM. At least, it needs to know where to route a message, e.g. TEE instance. It doesn't need to process or verify message content.

OTrP Agent returns TEE / TFW generated response messages to the caller. OTrP Agent isn't expected to handle any network connection with an application or TSM.

OTrP Agent only needs to return an OTrP Agent error message if the TEE is not reachable for some reason. Other errors are represented as response messages returned from the TEE which will then be passed to the TSM.

6.2. OTrP Agent and Global Platform TEE Client API

A Client Application may rely on the Global Platform (GP) TEE API for TA communication. OTrP may use the GP TEE Client API but it is internal to the OTrP implementation that converts given messages from TSM. More details can be found at [GPTEE].

6.3. OTrP Agent Implementation Consideration

A Provider should consider methods of distribution, scope and concurrency on device and runtime options when implementing an OTrP Agent. Several non-exhaustive options are discussed below. Providers are encouraged to take advantage of the latest communication and platform capabilities to offer the best user experience.

6.3.1. OTrP Agent Distribution

OTrP Agent installation is commonly carried out at OEM time. A user can dynamically download and install an OTrP Agent on-demand.

It is important to ensure a legitimate OTrP Agent is installed and used. If an OTrP Agent is compromised it may send rogue messages to TSM and TEE and introduce additional risks.

6.3.2. Number of OTrP Agents

We anticipate only one shared OTrP Agent instance in a device. The device's TEE vendor will most probably supply one OTrP Agent. Potentially we expect some open source.

With one shared OTrP Agent, the OTrP Agent provider is responsible to allow multiple TSMs and TEE providers to achieve interoperability. With a standard OTrP Agent interface, a TSM can implement its own SDK for its SP Client Applications to work with this OTrP Agent.

Multiple independent OTrP Agent providers can be used as long as they have standard interface to a Client Application or TSM SDK. Only one OTrP Agent is expected in a device.

Commented [DT37]: OEM is an actor not a time. Do you mean “at manufacturing time”?

Commented [DT38]: The only way to really “ensure” is would be for the agent to run inside the TEE. But above the agent is defined as being in the REE so you cannot “ensure” this. The key is to make it be not critical to ensure this, i.e., ensure that a rogue agent can only do DoS, nothing else. You can’t prevent DoS entirely since the network connection to the device can be untrusted.

OTrP ~~Protocol~~ MUST specify a standard way for applications to lookup the active OTrP Agent instance in a device.

TSM providers are generally expected to provide an SDK for SP applications to interact with the OTrP Agent for the TSM and TEE interaction.

6.3.3. OTrP Android Service Option

OTrP Agent can be a bound service in Android with a service registration ID that a Client Application can use. This option allows a Client Application not to depend on any OTrP Agent SDK or provider.

An OTrP Agent is responsible to detect and work with more than one TEE if a device has more than one. In this version, there is only one active TEE such that an OTrP Agent only needs to handle the active TEE.

6.4. OTrP Agent API for Client Applications

A Client Application shall be responsible for relaying messages between the OTrP agent and the TSM.

OTrP Agent APIs are defined below. An OTrP Agent in the form of an Android bound service can take this to be the functionality it provides via service call. The OTrP Agent implements this interface.

If a failure is occured during calling API, an error message described in "Common Errors" section (see Section 7.6) will be returned.

```
interface IOTrPAgentService {
  String processMessage(String tsmInMsg) throws OTrPAgentException;
  String getTAInformation(String spid, String taid)
     throws OTrPAgentException;
}

public class OTrPAgentException extends Throwable {
  private int errCode;
}
```

6.4.1. API processMessage

```
String processMessage(String tsmInMsg) throws OTrPAgentException;
```

Description

Commented [DT39]: This is the OTrP spec. You shouldn't have a MUST for yourself as an author. Just do it in the spec.

Commented [DT40]: Either remove this section or move into an appendix and cover multiple OS's. The body of the doc should not be OS-specific.

Commented [DT41]: The IETF does not define concrete APIs (i.e. APIs in specific programming languages). Either just remove all APIs or replace them with abstract (i.e. programming language agnostic) APIs like the GSSAPI RFC (RFC 2078) does.

A Client Application will use this method of the OTrP Agent in a device to pass OTrP messages from a TSM. The method is responsible for interation with the TEE and for forwarding the input message to the TEE. It also returns TEE generated response message back to the Client Application.

Commented [DT42]: integration?

Input

tsmInMsg - OTrP message generated in a TSM that is passed to this method from a Client Application.

Output

A TEE-generated OTrP response message (which may be a successful response or be a response message containing an error raised within the TEE) for the client application to forward to the TSM. In the event of the OTrP agent not being able to communicate with the TEE, a OTrPAgentException shall be thrown.

6.4.2. API getTAInformation

```
String getTAInformation(String spid, String taid)
   throws OTrPAgentException;
```

Description

A Client Application calls this method to query a TA's information. This method is carried out locally by the OTrP Agent without relying on a TSM if it has had the TEE SP AIK.

Input

spid - SP identifier of the TA

taid - the identifier of the TA

Output

The API returns TA signer and TSM signer certificate along with other metadata information about a TA.

The output is a JSON message that is generated by the TEE. It contains the following information:

* TSMID

* SP ID

* TA signer certificate

* TSM certificate

The message is signed with TEE SP AIK private key.

The Client Application is expected to consume the response as follows.

The Client Application gets signed TA metadata, in particular~~ly~~, the TA signer certificate. It is able to verify that the result is from device by checking signer against TEE SP AIK public key it gets in some earlier interaction with TSM.

If this is a new Client Application in the device that hasn't had TEE SP AIK public key for the response verification, the application can contact TSM first to do GetDeviceState, and the TSM will return TEE SP AIK public key to the app for this operation to proceed.

JSON Message

Commented [DT43]: This part looks like it's part of the protocol (meaning it appears over the wire). If so then it should not be in the same section as the API, which doesn't appear over the wire.

And for specifying JSON syntax, would be better to use a formal mechanism like JSON Schema or ABNF.

```
{
  "TAInformationTBS": {
    "taid": "<TA Identifier from the input>",
    "tsmid": "<TSM ID for the Security Domain where this TA
              resides>",
    "spid": "<The service provider identifier of this TA>",
    "signercert": "<The BASE64 encoded certificate data of the TA
                   binary application's signer certificate>",
    "signercacerts": [ // the full list of CA certificate chain
                       // including the root CA
    "cacert": "<The BASE64 encoded CA certificate data of the TA
                   binary application's signer certificate>"
    ],
    "tsmcert": "<The BASE64 encoded certificate data of the TSM that
                  manages this TA.>",
    "tsmcacerts": [ // the full list of CA certificate chain
                    // including the root CA
    "cacert":"<The BASE64 encoded CA certificate data of the TSM
                  that manages this TA>"
    ]
  }
}

{
  "TAInformation": {
      "payload": "<BASE64URL encoding of the TAInformationTBS
                  JSON above>",
      "protected": "<BASE64URL encoded signing algorithm>",
      "header": {
          "signer": {"<JWK definition of the TEE SP AIK public
                      key>"}
      },
      "signature": "<signature contents signed by TEE SP AIK private
                    key BASE64URL encoded>"
  }
}
```

A sample JWK public key representation refers to an example in RFC 7517 [RFC7517].

6.5. Sample End-to-End Client Application Flow

6.5.1. Case 1: A new Client App uses a TA

1. During the Client App installation time, the Client App calls TSM to initialize device preparation

Commented [DT44]: Add citation

Commented [DT45]: Missing close]

Commented [DT46]: Missing close]

Commented [DT47]: Add citation

A. The Client Application knows it wants to use a TA1 but the application doesn't know whether TA1 has been installed or not. It can use GP TEE Client API to check the existence of TA1 first. If it doesn't exist, it will contact TSM to initiate the TA1 installation. Note that TA1 could have been installed that is triggered by other Client Applications of the same service provider in the same device.

B. The Client Application sends the TSM the TA list that it depends on. The TSM will query a device for the Security Domains and TAs that have been installed, and instructs the device to install any dependent TAs that have not been installed.

C. In general, TSM has the latest information of TA list and their status in a device because all operations are instructed by TSM. TSM has such visibility because all Security Domain deletion and TA deletion are is managed by the TSM; the TSM could have stored the state when a TA is installed, updated and deleted. There is also the possibility that an update command is carried out inside TEE but a response is never received in TSM. There is also possibility that some manual local reset is done in a device that the TSM isn't aware of the changes.

2. The TSM generates message:a GetDeviceStateRequest message.

3. The Client Application passes the JSON message GetDeviceStateRequest to OTrP Agent API processMessage. The communication between a Client Application and OTrP Agent is up to the implementation of the OTrP Agent.

4. The OTrP Agent routes the message to the active TEE. Multiple TEE case: it is up to OTrP Agent to figure this out. This specification limits the support to only one active TEE, which is the typical case today.

5. The target active TEE processes the received OTrP message, and returns a JSON message GetDeviceStateResponse.

6. The OTrP Agent passes the GetDeviceStateResponse to the Client Application.

7. The Client Application sends the GetDeviceStateResponse to the TSM.

8. The TSM processes the GetDeviceStateResponse.

Commented [DT48]: Add citation

Commented [DT49]: Can't parse this sentence

A. Extract TEEspaik for the SP, signs TEEspaik with TSM signer key

B. Examine SD list and TA list

9. TSM continues to carry out other actions bas<u>ed</u>~~ing~~ on the need. The next call could be instructing the device to install a dependent TA.

A. Assume a dependent TA isn't in the device yet, the TSM may do the following:

~~B.~~

Create a<u>n</u> SD <u>in which </u>to install the TA by sending a ~~message~~ CreateSDRequest<u> message</u>. The message is sent back to the Client Application, and then <u>the </u>OTrP Agent and TEE to process.

Install a TA with a ~~message~~ InstallTARequest<u> message</u>.

<u>B</u>~~C~~. If a Client Application depends on multiple TAs, the Client Application should expect multiple round trips of the TA installation message exchanges.

10. At the last TSM and TEE operation, <u>the </u>TSM returns the signed TEE SP AIK public key to the application<u>.</u>

11. The Client Application shall store the TEEspaik for future loaded TA trust check purpose.

Commented [DT50]: Don't use "shall" in an *example*

12. If the TSM finds that this is a fresh device that does not have any SD for the SP yet, then the TSM may ~~move on to~~<u>next</u> create a<u>n</u> SD for the SP~~ next~~.

13. During Client Application installation, the application checks whether required Trusted Applications are already installed, which may have been provided by <u>the </u>TEE. If needed, it will contact its TSM service to determine whether the device is ready or install TA list that this application needs.

6.5.2. Case 2: A previously installed Client Application calls a TA

1. The Client Application checks the device readiness: (a) whether it has a TEE; (b) whether it has TA that it depends. It may happen that TSM has removed <u>the </u>TA this application depends on.

2. The Client App calls <u>the </u>OTrP Agent method "GetTAInformation"

3. The OTrP Agent queries the TEE to get TA information. If the given TA doesn't exist, an error is returned

4. The Client App parses the TAInformation message.

5. If the TA doesn't exist, the Client App calls its TSM to install the TA. If the TA exists, the Client App proceeds to call the TA.

7. OTrP Messages

The main OTrP Protocol component is the set of standard JSON messages created by a TSM to deliver device SD and TA management commands to a device, and device attestation and response messages created by a TEE to respond to TSM OTrP Messages.

An OTrP Message is designed to provide end-to-end security. It is always signed by its creator. In addition, an OTrP Message is typically encrypted such that only the targeted device TEE or TSM provider is able to decrypt and view the actual content.

7.1. Message Format

OTrP Messages use the JSON format for JSON's simple readability and moderate data size in comparison with alternative TLV and XML formats.

JSON Message security has developed JSON Web Signing and JSON Web Encryption standard in the IETF Workgroup JOSE, see JWS [RFC7515] and JWE [RFC7516]. The OTrP Messages in this protocol will leverage the basic JWS and JWE to handle JSON signing and encryption.

7.2. Message Naming Convention

For each TSM command "xyz"", OTrP ~~Protocol~~ uses the following naming convention to represent its raw message content and complete request and response messages:

Commented [DT51]: I think this word should be deleted. The “provider” is (to me) the software vendor. There’s no reason I see to require it to be readable by the vendor per se, only by the device itself.

Commented [DT52]: This is the section where I think it should allow CBOR derived from the JSON.

Purpose	Message Name	Example
Request to be signed	xyzTBSRequest	CreateSDTBSRequest
Request message	xyzRequest	CreateSDRequest
Response to be signed	xyzTBSResponse	CreateSDTBSResponse
Response message	xyzResponse	CreateSDResponse

7.3. Request and Response Message Template

An OTrP Request message uses the following format:

```
{
  "<name>TBSRequest": {
    <request message content>
  }
}
```

A corresponding OTrP Response message will be as follows.

```
{
  "<name>TBSResponse": {
    <response message content>
  }
}
```

7.4. Signed Request and Response Message Structure

A signed request message will generally include only one signature, and uses the flattened JWS JSON Serialization Syntax, see Section 7.2.2 in RFC7515 [RFC7515].

A general JWS object looks like the following.

```
{
  "payload": "<payload contents>",
  "protected": "<integrity-protected header contents>",
  "header": {
    <non-integrity-protected header contents>,
  },
  "signature": "<signature contents>"
}
```

OTrP signed messages only require~~s~~ the signing algorithm as the mandate header in the property "protected". The "non-integrity-protected header contents" is optional.

OTrP signed message will be given an explicit Request or Response property name. In other words, a signed Request or Response uses the following template.

A general JWS object looks like the following.

```
{
  "<name>[Request | Response]": {
    <JWS Message of <name>TBS[Request | Response]
  }
}
```

With the standard JWS message format, a signed OTrP Message looks like the following.

```
{
  "<name>[Request | Response]": {
    "payload": "<payload contents of <name>TBS[Request | Response]>",
    "protected": "<integrity-protected header contents>",
    "header":  <non-integrity-protected header contents>,
    "signature": "<signature contents>"
  }
}
```

The top element " <name>[Signed][Request | Response]" cannot be fully trusted to match the content because it doesn't participate in the signature generation. However, a recipient can always match it with the value associated with the property "payload". It purely serves to provide a quick reference for reading and method invocation.

Furthermore, most properties in an unsigned OTrP messages are encrypted to provide end-to-end confidentiality. The only OTrP message that isn't encrypted is the initial device query message that asks for the device state information.

Thus a typical OTrP Message consists of an encrypted and then signed JSON message. Some transaction data such as transaction ID and TEE information may need to be exposed to the OTrP Agent for routing purpose. Such information is excluded from JSON encryption. The device's signer certificate itself is encrypted. The overall final message is a standard signed JSON message.

As required by JSW/JWE, those JWE and JWS related elements will be BASE64URL encoded. Other binary data elements specific to the OTrP specification are BASE64 encoded. This specification will identify elements that should be BASE64 and those elements that are to be BASE64URL encoded.

Commented [DT53]: But it doesn't yet?

7.4.1. Identifying signing and Encryption keys for JWS/JWE messaging

JWS and JWE messaging allow various options for identifying the signing and encryption keys, for example, it allows optional elements including "x5c", "x5t" and "kid" in the header to cover various possibilities.

~~In order t~~To protect privacy, it is important that the device's certificate is released only to a trusted TSM, and that it is encrypted. The TSM will need to know the device certificate, but untrusted parties must not be able to get the device certificate. All OTrP messaging conversations between a TSM and device begin with GetDeviceStateRequest / GetDeviceStateResponse. These messages have elements built into them to exchange signing certificates, described in the "Detailed Message Specification" section. Any subsequent messages in the conversation that follow on from this ~~are~~ implicitly use~~ing~~ the same certificates for signing/encryption, and as a result the certificates or references may be om~~m~~itted in those subsequent messages.

Commented [DT54]: Reference it by section number.

In other words, the signing key identifier in the use of JWS and JWE here may be absent in the subsequent messages after the initial GetDeviceState query.

This has an implication on the TEE and TSM implementations: they have to cache the signer certificates for the subsequent message signature validation in the session. It may be easier for a TSM service to cache transaction session information but not so for a TEE in a device. A TSM should check a device's capability to decide whether it should include its TSM signer certificate and OCSP data in each subsequent request message. The device's caching capability is reported in GetDeviceStateResponse signerreq parameter.

Commented [DT55]: Be specific: *How* should it decide based on the signerreq parameter?

7.5. JSON Signing and Encryption Algorithms

The OTrP JSON signing algorithm shall use SHA256 or a stronger hash method with respective key type. JSON Web Algorithm RS256 or ES256 [RFC7518] SHALL be used for RSA with SHA256 and ECDSA with SHA256. If RSA with SHA256 is used, the JSON web algorithm representation is as follows.

```
{"alg":"RS256"}
```

The (BASE64URL encoded) "protected" header property in a signed message looks like the following:

```
"protected":"eyJhbGciOiJSUzI1NiJ9"
```

If ECSDA with P-256 curve and SHA256 are used for signing, the JSON signing algorithm representation is as follows.

```
{"alg":"ES256"}
```

The value for the "protected" field will be the following.

```
eyJhbGciOiJFUzI1NiJ9
```

Thus, a common OTrP signed message with ES256 looks like the following.

```
{
  "payload": "<payload contents>",
  "protected": "eyJhbGciOiJFUzI1NiJ9",
  "signature": "<signature contents>"
}
```

The OTrP JSON message encryption algorithm should use one of the supported algorithms defined in the later chapter of this document. JSON encryption uses a symmetric key as its "Content Encryption Key (CEK)". This CEK is encrypted or wrapped by a recipient's key. The OTrP recipient typically has an asymmetric key pair. Therefore, the CEK will be encrypted by the recipient's public key.

Commented [DT56]: SHOULD?

Symmetric encryption shall use the following algorithm.

Commented [DT57]: This is apparently not crypto-agile then

```
{"enc":"A128CBC-HS256"}
```

This algorithm represents encryption with AES 128 in CBC mode with HMAC SHA 256 for integrity. The value of the property "protected" in a JWE message will be

```
eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0
```

An encrypted JSON message looks like the following.

```
{
  "protected": "eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
  –"recipients": [
      {
          "header": {
              "alg": "<RSA1_5 etc.>"
          },
          "encrypted_key": "<encrypted value of CEK>"
      }
  ],
  "iv": "<BASE64URL encoded IV data>",
  "ciphertext": "<Encrypted data over the JSON plaintext
                (BASE64URL)>",
  "tag": "<JWE authentication tag (BASE64URL)>"
}
```

OTrP doesn't use JWE AAD (Additional Authenticated Data) because each message is always signed after the message is encrypted.

7.5.1. Supported JSON Signing Algorithms

The following JSON signature algorithm ~~are~~ is mandatory support in the TEE and TSM:

- o RS256

ES256 is optional to support.

7.5.2. Support JSON Encryption Algorithms

The following JSON authenticated encryption algorithm is mandatory support in TEE and TSM.

- o A128CBC-HS256

A256CBC-HS512 is optional to support.

7.5.3. Supported JSON Key Management Algorithms

The following JSON key management algorithm is mandatory support in TEE and TSM.

- o RSA1_5

ECDH-ES+A128KW and ECDH-ES+A256KW are optional to support.

7.6. Common Errors

An OTrP Response message typically needs to report the operation status and error causes if an operation fails. The following JSON message elements should be used across all OTrP Messages.

```
"status": "pass | fail"

 "reason": {
     "error-code": "<error code if there is any>",
     "error-message": "<error message>"
  }


"ver": "<version string>"
```

7.7. OTrP Message List

The following table lists the OTrP commands and therefore corresponding Request and Response messages defined in this specification. Additional messages may be added in the future when new task messages are needed.

GetDeviceState -
    A TSM queries a device's current state with a message GetDeviceStateRequest. A device TEE will report its version, its FW version, and list of all SDs and TAs in the device that is managed by the requesting TSM. TSM may determine whether the device is trustworthy and decide to carry out additional commands according to the response from this query.

CreateSD -
    A TSM instructs a device TEE to create an SD for an SP. The recipient TEE will check whether the requesting TSM is trustworthy.

UpdateSD -
    A TSM instructs a device TEE to update an existing SD. A typical update need comes from SP certificate change, TSM certificate change and so on. The recipient TEE will verify whether the TSM is trustworthy and owns the SD.

DeleteSD -
    A TSM instructs a device TEE to delete an existing SD. A TEE conditionally deletes TAs loaded in the SD according to a request parameter. A SD cannot be deleted until all TAs in this SD are deleted. If this is the last SD for an SP, the TEE can also delete the TEE SP AIK key for this SP.

Commented [DT58]: MAY? SHOULD? MUST?

InstallTA -
A TSM instructs a device to install a TA into an SD for an SP. The TEE in a device will check whether the TSM and TA are trustworthy.

UpdateTA -
A TSM instructs a device to update a TA into an SD for an SP. The change may commonly be a bug fix for a previously installed TA.

DeleteTA -
A TSM instructs a device to delete a TA. The TEE in a device will check whether the TSM and TA are trustworthy.

7.8. OTrP Request Message Routing Rules

For each command that a TSM wants to send to a device, the TSM generates a request message. This is typically triggered by a Client Application that uses the TSM. The Client Application initiates contact with the TSM and receives TSM OTrP Request messages according to the TSM's implementation. The Client Application forwards the OTrP message to an OTrP Agent in the device, which in turn sends the message to the active TEE in the device.

The current version of this specification assumes that each device has only one active TEE, and the OTrP Agent is responsible to connect to the active TEE. This is the case today with devices in the market.

~~Upon~~ When the TEE respond~~sing~~ ~~with~~ to a request, the OTrP Agent gets the OTrP response message~~s~~ back to the Client Application that sent~~ds~~ the request. In case the target TEE fails to respond to the request, the OTrP Agent will be responsible to generate an error message to reply to the Client Application. The Client Application forwards any data it received to its TSM.

7.8.1. SP Anonymous Attestation Key (SP AIK)

When the first new Security Domain is created in a TEE for an SP, a new key pair is generated and associated with this SP. This key pair is used for future device attestation to the service provider instead of using the device's TEE key pair.

8. Detailed Messages Specification

For each message in the following sections all JSON elements are mandatory if ~~it isn't~~not explicitly indicated as optional.

8.1. GetDeviceState

This is the first command that a TSM will ~~query~~ send to a device. This command is triggered when an SP's Client Application contacts its TSM to check whether the underlying device is ready for TA operations.

This command queries a device's current TEE state. A device TEE will report its version, its FW version, and list of all SDs and TAs in the device that is managed by the requesting TSM. The TSM may determine whether the device is trustworthy and decide to carry out additional commands according to the response from this query.

The request message of this command is signed by the TSM. The response message from the TEE is encrypted. A random message encryption key (MK) is generated by TEE, and this encrypted key is encrypted by the ~~receiving~~ TSM's public key such that only the TSM ~~who~~ that sent the request is able to decrypt and view the response message.

Commented [DT59]: … but not encrypted?

8.1.1. GetDeviceStateRequest message

```
{
   "GetDeviceStateTBSRequest": {
      "ver": "1.0",
      "rid": "<Unique request ID>",
      "tid": "<transaction ID>",
      "ocspdat": "<OCSP stapling data of TSM certificate>",
      "icaocspdat": "<OCSP stapling data for TSM CA certificates>",
      "supportedsigalgs": "<comma separated signing algorithms>"
    }
}
```

Commented [DT60]: Why comma separated? This is harder to parse, why not just use an array?

The request message consists of the following data elements:

ver - version of the message format

rid - a unique request ID generated by the TSM

tid - a unique transaction ID to trace request and response. This can be from a prior transaction's tid field, and can be used in ~~the~~ subsequent message exchanges in this TSM session. The combination of rid and tid should be made unique.

Commented [DT61]: MUST? SHOULD?

ocspdat - OCSP stapling data for the TSM certificate. The TSM provides OCSP data such that a recipient TEE can validate the ~~validity of the~~ TSM certificate without making its own external OCSP service call. This is a mandatorye field.

Commented [DT62]: Use of JSON Schema or ABNF would make this sentence redundant icaocspdat - OCSP stapling data for the intermediate CA certificates of the TSM certificate up to the root. A TEE ~~side~~ can cache CA OCSP data such that this value isn't needed in each call.

Commented [DT63]: MAY?

supportedsigalgs - an optional property to list the signing algorithms that the TSM is able to support. A recipient TEE should choose an algorithm in this list to sign its response message if this property is present in a request.

Commented [DT64]: SHOULD? MUST?

Commented [DT65]: And what if it's not present?

The final request message is JSON signed message of the above raw JSON data with TSM's certificate.

```
{
  "GetDeviceStateRequest": {
    "payload": "<BASE64URL encoding of the GetDeviceStateTBSRequest
               JSON above>",
    "protected": "<BASE64URL encoded signing algorithm>",
    "header": {
        "x5c": "<BASE64 encoded TSM certificate chain up to the
                root CA certificate>"
    },
    "signature":"<signature contents signed by TSM private key>"
  }
}
```

The signing algorithm should use SHA256 with respective key type. The mandatory algorithm support is the RSA signing algorithm. The signer header "x5c" is used to include the TSM signer certificate up to the root CA certificate.

Commented [DT66]: SHOULD?

8.1.2. Request processing requirements at a TEE

Upon receiving a request message GetDeviceStateRequest at a TEE, the TEE must validate a request:

Commented [DT67]: MUST

1. Validate JSON message signing

Commented [DT68]: And do what if validation fails?

2. Validate that the request TSM certificate is chained to a trusted CA that the TEE embeds as its trust anchor.

   * Cache the CA OCSP stapling data and certificate revocation check status for other subsequent requests.

   * A TEE can use its own clock time for the OCSP stapling data validation.

3. Collect Firmware signed data

* This is a capability in ARM architecture that allows a TEE to query Firmware to get FW signed data.

Commented [DT69]: Delete or replace with vendor-agnostic or multi-vendor phrasing.

4. Collect SD information for the SD owned by this TSM

8.1.3. Firmware signed data

Firmware isn't expected to process or produce JSON data. It is expected to just sign some raw bytes of data.

The data to be signed by TFW key needs be some unique random data each time. The (UTF-8 encoded) "tid" value from the GetDeviceStateTBSRequest shall be signed by the firmware. The TSM isn't expected to parse TFW data except the signature validation and signer trust path validation.

It is possible that a TEE can get some valid TFW signed data from another device. This is part of the TEE trust assumption where the TSM will trust the TFW data supplied by the TEE. The TFW trust is more concerned by TEE than a TSM where a TEE needs to ensure that the underlying device firmware is trustworthy.

Commented [DT70]: If I understand correctly this sounds like a poor assumption.

Commented [DT71]: I can't parse this sentence.

```
TfwData: {
     "tbs": "<TFW to be signed data, BASE64 encoded>",
     "cert": "<BASE64 encoded TFW certificate>",
     "sigalg": "Signing method",
     "sig": "<Tfw TFW signed data,  BASE64 encoded>"
}
```

It is expected that FW use ~~a~~ standard signature methods for maximal interoperability with TSM providers. The mandatory support list of signing algorithm is RSA with SHA256.

The JSON object above is constructed by a TEE with data returned from FW. It isn't a standard JSON signed object. The signer information and data to be signed must be specially processed by a TSM according to the definition given here. The data to be signed is the raw data.

8.1.3.1. Supported Firmware Signature Methods

TSM providers shall support the following signature methods. A firmware provider can choose one of the methods in signature generation.

- RSA with SHA256
- ECDSA with SHA 256

The value of "sigalg" in the TfwData JSON message should use one of the following:

o RS256

o ES256

8.1.4. Post Conditions

Upon successful request validation, the TEE information is collected. There is no change in the TEE in the device.

The response message shall be encrypted where the encryption key shall be a symmetric key that is wrapped by TSM's public key. The JSON Content Encryption Key (CEK) is used for this purpose.

8.1.5. GetDeviceStateResponse message

The message has the following structure.

```
   {
     "GetDeviceTEEStateTBSResponse": {
         "ver": "1.0",
         "status": "pass | fail",
         "rid": "<the request ID from the request message>",
         "tid": "<the transaction ID from the request message>",
         "signerreq": "true | false about whether TSM needs to send
                       signer data again in subsequent messages",
         "edsi": "<Encrypted JSON dsi DSI information>"
     }
  }
```

where

signerreq - true if the TSM should send its signer certificate and OCSP data again in the subsequent messages. The value may be "false" if the TEE caches the TSM's signer certificate and OCSP status.

rid - the request ID from the request message

tid - the tid from the request message

edsi - the main data element whose value is JSON encrypted message over the following Device State Information (DSI).

The Device State Information (DSI) message consists of the following.

Commented [DT72]: SHOULD? MUST?

Commented [DT73]: This isn't as compressable as a Boolean. Do you expect other status values in the future?

Commented [DT74]: True and false are legal JSON boolean values, why do you need strings?

```
{
    "dsi": {
        "tfwdata": {
            "tbs": "<TFW to be signed data is the tid>"
            "cert": "<BASE64 encoded TFW certificate>",
            "sigalg": "Signing method",
            "sig": "<Tfw TFW signed data, -BASE64 encoded>"
        },
        "tee": {
            "name": "<TEE name>",
            "ver": "<TEE version>",
            "cert": "<BASE64 encoded TEE cert>",
            "cacert": "<JSON array value of CA certificates up to
                        the root CA>",
            "sdlist": {
                "cnt": "<Number of SD owned by this TSM>",
                "sd": [
                    {
                        "name": "<SD name>",
                        "spid": "<SP owner ID of this SD>",
                        "talist": [
                          {
                             "taid": "<TA application identifier>",
                             "taname": "<TA application friendly
                                       name>" // optional
                          }
                        ]
                    }
                ]
            },
            "teeaiklist": [
                {
                    "spaik": "<SP AIK public key, BASE64 encoded>",
                    "spaiktype": "<RSA | ECC>",
                    "spid": "<sp id>"
                }
            ]
        }
    }
}
```

The encrypted JSON message looks like the following.

```
{
    "protected": "<BASE64URL encoding of encryption algorithm header
                   JSON data>",
    "recipients": [
        {
            "header": {
                "alg": "RSA1_5"
            },
            "encrypted_key": "<encrypted value of CEK>"
        }
    ],
    "iv": "<BASE64URL encoded IV data>",
    "ciphertext": "<Encrypted data over the JSON object of dsi
                    (BASE64URL)>",
    "tag": "<JWE authentication tag (BASE64URL)>"
}
```

Assume we encrypt plaintext with AES 128 in CBC mode with HMAC SHA 256 for integrity, the encryption algorithm header is:

```
{"enc":"A128CBC-HS256"}
```

Commented [DT75]: Can't parse this sentence

The value of the property "protected" in the above JWE message will be

```
eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0
```

In other words, the above message looks like the following:

```
{
    "protected": "eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
     "recipients": [
        {
            "header": {
                "alg": "RSA1_5"
            },
            "encrypted_key": "<encrypted value of CEK>"
        }
    ],
    "iv": "<BASE64URL encoded IV data>",
    "ciphertext": "<Encrypted data over the JSON object of dsi
                    (BASE64URL)>",
    "tag": "<JWE authentication tag (BASE64URL)>"
}
```

The full response message looks like the following:

```
{
  "GetDeviceTEEStateTBSResponse": {
    "ver": "1.0",
    "status": "pass | fail",
    "rid": "<the request ID from the request message>",
    "tid": "<the transaction ID from the request message>",
    "signerreq": "true | false",
    "edsi": {
      "protected": "<BASE64URL encoding of encryption algorithm
                    header JSON data>",
      "recipients": [
        {
          "header": {
            "alg": "RSA1_5"
          },
          "encrypted_key": "<encrypted value of CEK>"
        }
      ],
      "iv": "<BASE64URL encoded IV data>",
      "ciphertext": "<Encrypted data over the JSON object of dsi
                     (BASE64URL)>",
      "tag": "<JWE authentication tag (BASE64URL)>"
    }
  }
}
```

The CEK will be encrypted by the TSM public key in the device. The TEE signed message has the following structure.

```
{
  "GetDeviceTEEStateResponse": {
    "payload": "<BASE64URL encoding of the JSON message
                GetDeviceTEEStateTBSResponse>",
    "protected": "<BASE64URL encoding of signing algorithm>",
    "signature": "<BASE64URL encoding of the signature value>"
  }
}
```

The signing algorithm shall use SHA256 with respective key type, see Section ~~Section~~ 7.5.1.

The final ~~response message~~ GetDeviceStateResponse response message consists of an array of TEE responses. A typical device will have only one active TEE. An OTrP Agent is responsible to collect TEE response for all active TEEs in the future.

Commented [DT76]: Earlier in the draft it said this version of the doc assumes only one active TEE so you don't need this text if that's the case.

```
{
    "GetDeviceStateResponse": [ // JSON array
       {"GetDeviceTEEStateResponse": ...},
       ...
       {"GetDeviceTEEStateResponse": ...}
    ]
}
```

8.1.6. Error Conditions

An error may occur if a request isn't valid or the TEE runs into some error. The list of possible error conditions is the following.

ERR_REQUEST_INVALID The TEE meets the following conditions with a request message: (1) The request from a TSM has an invalid message structure; mandatory information is absent in the message~~.~~; or an undefined member or structure is included. (2) TEE fails to verify the signature of the message or fails to decrypt its contents. (3) etc.

Commented [DT77]: Not sure how an implementer is supposed to interpret this. Suggest deleting.

ERR_UNSUPPORTED_MSG_VERSION The TEE receives ~~the ~~a version of message that the TEE can't deal with.

ERR_UNSUPPORTED_CRYPTO_ALG The TEE receives a request message encoded with cryptographic algorithms that the TEE doesn't support.

ERR_TFW_NOT_TRUSTED The TEE ~~may ~~considers the underlying device firmware be not trustworthy.

ERR_TSM_NOT_TRUSTED The TEE needs to make sure whether the TSM is trustworthy by checking the validity of the TSM certificate and OCSP stapling data and so on. If the TEE finds the TSM is not reliable, it ~~may~~ returns this error code.

ERR_TEE_FAIL The TEE failed~~s~~ to respond to a TSM request. The OTrP Agent will construct an error message in responsese~~ding~~ to the TSM's request. ~~And a~~Also if the TEE fails to process a request because of its internal error, it will return this error code.

Commented [DT78]: I think this is a mistake. The former case is untrusted, but this case can be fully trusted. As such, they should be two separate error codes, and the TEE's should be signed.

The response message will look like the following if the TEE signing can work to sign the error response message.

```
{
    "GetDeviceTEEStateTBSResponse": {
        "ver": "1.0",
        "status": "fail",
        "rid": "<the request ID from the request message>",
        "tid": "<the transaction ID from the request message>",
        "reason": {"error-code": "<error code>"}
        "supportedsigalgs": "<signature algorithms TEE supports>"
    }
}
```

where

supportedsigalgs - an optional property to list the JWS signing algorithms that the active TEE supports. When a TSM sends a signed message that the TEE isn't able to validate, it can include signature algorithms that it is able to consume in this status report. A TSM can generate a new request message to retry the management task with a TEE-~~–~~supported signing algorithm.

If TEE isn't able to sign an error message, a general error message should be returned.

Commented [DT79]: When would it not be able to?

8.1.7. TSM Processing Requirements

Upon receiving a ~~message of the type~~ GetDeviceStateResponse message~~at a TSM~~, the TSM should validate the following.

Commented [DT80]: MUST?

- Parse to get list of GetDeviceTEEStateResponse JSON objects

- Parse the JSON "payload" property and decrypt the JSON element "edsi".

~~o~~ The decrypted message contains the TEE signer certificate.

- Validate the GetDeviceTEEStateResponse JSON signature. The signer certificate is extracted from the decrypted message in the last step.

- Extract TEE information and check it against its TEE acceptance policy.

- Extract the TFW signed element, and check the signer and data integration against its TFW policy.

- Check the SD list and TA list and prepare for a subsequent command such as "CreateSD" if it needs to have a new SD for a SP.

8.2. Security Domain Management

8.2.1. CreateSD

This command is typically preceded with a GetDeviceState command that has acquired the device information of the target device by the TSM. The TSM sends such a command to instruct a TEE to create a new Security Domain for a SP.

A TSM sends an OTrP Request message CreateSDRequest Request message to a device TEE to create a Security Domain for a SP. Such a request is signed by the TSM where the TSM signer may or may not be the same as the SP's TA signer certificate. The resulting SD is associated with two identifiers for future management:

- o TSM as the owner. The owner identifier is a registered unique TSM ID that is stored in the TSM certificate.

- o SP identified by its TA signer certificate as the authorization. A TSM can add more than one SP certificates to an SD.

A Trusted Application that is signed by a matching SP signer certificate for an SD is eligible to be installed into that SD. The TA installation into an SD by a subsequent InstallTARequest message may be instructed from a TSM or a Client Application.

Commented [DT81]: But the client app is not trusted since it's in the REE, so I don't follow.

8.2.1.1. CreateSDRequest Message

The request message for CreateSD has the following JSON format.

```
{
   "CreateSDTBSRequest": {
     "ver": "1.0",
     "rid": "<unique request ID>",
     "tid": "<transaction ID>", // this may be from prior message
     "tee": "<TEE routing name from the DSI for the SD's target>",
     "nextdsi": "true | false",
     "dsihash": "<hash of DSI returned in the prior query>",
     "content": ENCRYPTED { // this piece of JSON data will be
                         // encrypted
           "spid": "<SP ID value>",
        "sdname": "<SD name for the domain to be created>",
        "spcert": "<BASE64 encoded SP certificate>",
        "tsmid": "<An identifiable attribute of the TSM
                   certificate>",
        "did": "<SHA256 hash of the TEE cert>"
     }
   }
}
```

In the message,

rid - A unique value to identify this request

tid - A unique value to identify this transaction. It can have the same value for the tid in the preceding GetDeviceStateRequest.

tee - TEE ID returned from the previous ~~response~~ GetDeviceStateResponse.

nextdsi - Indicates whether the up--to--date Device State Information (DSI) ~~should~~ is to be returned in the response to this request.

dsihash - The BASE64--encoded SHA256 hash value of the DSI data returned in the prior TSM operation with this target TEE. This value is always included such that a receiving TEE can check whether the device state has changed since its last query. It helps enforce SD update order in the right sequence without accidently overwritinge an update that was done simultaneously.

content - The "content" is a JSON encrypted message that includes actual input for the SD creation. The encryption key is TSMmk that is encrypted by the target TEE's public key. The entire message is signed by the TSM private key TSMpriv. A separate TSMmk isn't used in the latest specification because JSON encryption will use a content encryption key for exactly the same purpose.

spid - A unique id assigned by the TSM for its SP. It should be unique within a TSM namespace.

sdname - a name unique to the SP. TSM should ensure it is unique for each SP.

spcert - The SP's TA signer certificate is included in the request. This certificate will be stored by the device TEE ~~and~~ which uses it to check against TA installation. Only if a TA is signed by a matching spcert associated with an SD will the TA ~~will~~ be installed into the SD.

tsmid - SD owner claim by TSM - An SD owned by a TSM will be associated with a trusted identifier defined as an attribute in the signer TSM certificate. The TEE ~~will be~~is responsible to assign this ID to the SD. The TSM certificate attribute for this attribute TSMID must be vetted by the TSM signer issuing CA. With this trusted identifier, the SD query at TEE can be fast upon TSM signer verification.

did - The SHA256 hash of the binary~~-~~encoded device TEE certificate. The encryption key CEK will be encrypted the recipient TEE's public key. This hash value in the "did" property allows the recipient TEE to check whether it is the expected target to receive such a request. If this isn't given, an OTrP message for device 2 could be sent to device 1. It is optional for the TEE to check because the successful decryption of the request message with this device's TEE private key already proves it is the target. This explicit hash value makes the protocol not dependent on message encryption method in future.

Following is the OTrP message template;~~,~~ the full request is signed message over the CreateSDTBSRequest as follows.

```
{
    "CreateSDRequest": {
        "payload": "<CreateSDTBSRequest JSON above>",
        "protected": "<integrity-protected header contents>",
        "header": ~~-~~<non-integrity-protected header contents>,
        "signature": "<signature contents signed by TSM private key>"
    }
}
```

The TSM signer certificate is included in the "header" property.

Commented [DT82]: MUST?

Commented [DT83]: Can't parse this sentence

8.2.1.2. Request processing requirements at a TEE

Upon receiving a ~~request message~~ CreateSDRequest request message at a TEE, the TEE must ~~validate a request~~do the following:

1. Validate the JSON request message as follows

   * Validate JSON message signing.

   * Validate that the request TSM certificate is chained to a trusted CA that the TEE embeds as its trust anchor.

   * Compare dsihash with its current state to make sure nothing has changed since this request was sent.

   * Decrypt to get the plaintext of the content: (a) spid, (b) sd name, (c) did

   * Check that a SPID is supplied.

   * spcert check: check it is a valid certificate (signature and format verification only)

   * Check "did" is the SHA256 hash of its TEEcert BER raw binary data

   * Check whether the requested SD already exists for the SP

   * Check that the TSMID in the request matches TSM certificate's TSM ID attribute

2. Create action

   * Create an SD for the SP with the given name

   * Assign the TSMID from the TSMCert to this SD

   * Assign the SPID and SPCert to this SD

   * Check whether a TEE SP AIK keypair already exists for the given SP ID

   * Create TEE SP AIK keypair if it doesn't exist for the given SP ID

   * Generate new DSI data if the request asks for updated DSI

3. Construct a CreateSDResponse message

Commented [DT84]: MUST

Commented [DT85]: And do what if validation fails? Presumably skip step 2 but do the others

Commented [DT86]: If the request was valid, …

* Create raw content

  + Operation status

  + "did" or full signer certificate information,

  + TEE SP AIK public key if DSI isn't going to be included

  + Updated DSI data if requested if the request asks for it

* The response message is encrypted with the same JWE CEK of the request without recreating a new content encryption key.

* The encrypted message is signed with TEEpriv. The signer information ("did" or TEEcert) is encrypted.

4. Deliver the response message. (a) The OTrP Agent returns this to the Client app; (b) The Client app passes this back to the TSM.

5. TSM process. (a) The TSM processes the response message; (b) The TSM can look up signer certificate from the device ID "did".

If a request is illegitimate or signature doesn't pass, a "status" property in the response will indicate the error code and cause.

8.2.1.3. CreateSDResponse Message

The response message for a CreateSDRequest contains the following content.

```
{
  "CreateSDTBSResponse": {
    "ver": "1.0",
    "status": "<operation result>",
    "rid": "<the request ID received>",
    "tid": "<the transaction ID received>",
    "content": ENCRYPTED {
      "reason": "<failure reason detail>", // optional
      "did": "<the device id received from the request>",
      "sdname": "<SD name for the domain created>",
      "teespaik": "<TEE SP AIK public key, BASE64 encoded>",
      "dsi": "<Updated TEE state, including all SDs owned by
        this TSM>"
    }
  }
}
```

In the response message, the following fields MUST be supplied.

did -   The SHA256 hash of the device TEE certificate. This shows the device ID explicitly to the receiving TSM.

teespaik -   The newly generated SP AIK public key for the given SP. This is an optional value if the device has had another domain for the SP that has triggered TEE SP AIK keypair for this specific SP.

There is a possible extreme error case where the TEE isn't reachable or the TEE final response generation itself fails. In this case, the TSM ~~should~~might still receive a response from the OTrP Agent if~~.~~ the OTrP Agent is able to detect such error from TEE. In this case, a general error response message should be returned, assuming the OTrP Agent even doesn't know any content and information about the request message.

In other words, the TSM should expect to receive a TEE successfully signed JSON message, or a general "status" message.

```
{
  "CreateSDResponse": {
    "payload": "<CreateSDTBSResponse JSON above>",
    "protected": {
       "<BASE64URL of signing algorithm>"
    },
    "signature": "<signature contents signed by the TEE device private
                  key (BASE64URL)>"
  }
}
```

A response message type "status" will be returned when the TEE totally fails to respond. The OTrP Agent is responsible to create this message.

```
{
  "status": {
     "result": "fail",
     "error-code": "ERR_TEE_UNKNOWN",
     "error-message": "TEE fails to respond"
  }
}
```

8.2.1.4. Error Conditions

An error ~~may~~ might occur if a request isn't valid or the TEE runs into some error. The list of possible errors are the following. Refer to ~~section~~ the Error Code List (Section 14.1) for detailed causes and actions.

ERR_REQUEST_INVALID

Commented [DT87]: By whom? The OTrP Agent?

Commented [DT88]: But it has to be able to deal with not receiving it either due to lack of connectivity to the device, or due to a bad or broken client app. So it cannot have a strong expectation.

ERR_UNSUPPORTED_MSG_VERSION

ERR_UNSUPPORTED_CRYPTO_ALG

ERR_DEV_STATE_MISMATCH

ERR_SD_ALREADY_EXIST

ERR_SD_NOT_FOUND

ERR_SPCERT_INVALID

ERR_TEE_FAIL

ERR_TEE_UNKNOWN

ERR_TSM_NOT_AUTHORIZED

ERR_TSM_NOT_TRUSTED

8.2.2. UpdateSD

This TSM-~~-~~initiated command can update an SP's SD that it manages for any of the following needs:~~.~~ (a) Update SP signer certificate; (b) Add an SP signer certificate when an SP uses multiple to sign TA binaries~~y~~; (c) Update an SP ID.

The TSM presents the proof of the SD ownership to the TEE, and includes related information in its signed message. The entire request is also encrypted for ~~the~~ end-to-end confidentiality.

8.2.2.1. UpdateSDRequest Message

The ~~request message for~~ UpdateSD request message has the following JSON format.

```
{
   "UpdateSDTBSRequest": {
     "ver": "1.0",
     "rid": "<unique request ID>",
     "tid": "<transaction ID>", // this may be from prior message
     "tee": "<TEE routing name from the DSI for the SD's target>",
     "nextdsi": "true | false",
     "dsihash": "<hash of DSI returned in the prior query>",
     "content": ENCRYPTED { // this piece of JSON will be encrypted
       "tsmid": "<TSMID associated with this SD>",
       "spid": "<SP ID>",
       "sdname": "<SD name for the domain to be updated>",
       "changes": {
         "newsdname": "<Change the SD name to this new name>",
                       // Optional
         "newspid": "<Change SP ID of the domain to this new value>",
                       // Optional
         "spcert": ["<BASE64 encoded new SP signer cert to be added>"],
                       // Optional
         "deloldspcert": ["<The SHA256 hex value of an old SP cert
                    assigned into this SD that should be deleted >"],
                       // Optional
         "renewteespaik": "true | false"
         }
     }
   }
}
```

Commented [DT89]: Why not boolean?

Commented [DT90]: Why not boolean?

In the message,

rid - A unique value to identify this request

tid - A unique value to identify this transaction. It can have the same value ~~for~~ as the tid in the preceding GetDeviceStateRequest.

tee - TEE ID returned from the previous ~~response~~ GetDeviceStateResponse

nextdsi - Indicates whether the up-~~-~~to-~~-~~date Device State Information (DSI) ~~should~~ is to be returned in the response to this request.

dsihash - The BASE64=~~-~~encoded SHA256 hash value of the DSI data returned in the prior TSM operation with this target TEE. This value is always included such that a receiving TEE can check whether the device state has changed since its last query. It helps enforce SD update order in the right sequence without accidently overwrit~~e~~inge an update that was done simultaneously.

content - The "content" is a JSON encrypted message that includes actual input for the SD update. The standard JSON content encryption key (CEK) is used, and the CEK is encrypted by the target TEE's public key.

tsmid - SD owner claim by TSM - An SD owned by a TSM will be associated with a trusted identifier defined as an attribute in the signer TSM certificate.

spid - the identifier of the SP whose SD will be updated. This value is still needed because the SD name is considered unique only within an SP~~ only~~.

sdname - the name of the target SD to be updated.

changes - its content consists of changes that ~~should~~ are to be updated in the given SD.

newsdname - the new name of the target SD to be assigned if this value is present.

newspid - the new SP ID of the target SD to be assigned if this value is present.

spcert - a new TA signer certificate of this SP to be added to the SD if this is present.

deloldspcert - an SP certificate assigned into the SD ~~should~~ is to be deleted if this is present. The value is the SHA256 fingerprint of the old SP certificate.

renewteespaik - the value should be 'true' or 'false'. If it is present and the value is 'true', the TEE should regenerate TEE SP AIK for this SD's owner SP. The newly generated TEE SP AIK for the SP must be returned in the response message of this request. If there ~~are~~ is more than one SD for the SP, a new SPID for one of the domains will always trigger a new teespaik generation as if a new SP ~~is~~were introduced to the TEE.

Commented [DT91]: MUST?

Following the OTrP message template, the full request is signed message over the UpdateSDTBSRequest as follows.

```
{
  "UpdateSDRequest": {
    "payload": "<UpdateSDTBSRequest JSON above>",
    "protected": "<integrity-protected header contents>",
    "header": <non-integrity-protected header contents>,
    "signature": "<signature contents signed by TSM private key>"
  }
}
```

Commented [DT92]: In quotes?

TSM signer certificate is included in the "header" property.

8.2.2.2. Request processing requirements at a TEE

Upon receiving a request message UpdateSDRequest at a TEE, the TEE must validate a request:

1. Validate the JSON request message

   * Validate JSON message signing

   * Validate that the request TSM certificate is chained to a trusted CA that the TEE embeds as its trust anchor. The TSM certificate status check is generally not needed any more in this request. The prior request should have validated the TSM certificate's revocation status.

   * Compare dsihash with the TEE cached last response DSI data to this TSM

   * Decrypt to get the plaintext of the content

   * Check that the target SD name is supplied

   * Check whether the requested SD exists

   * Check that the TSM owns this TSM by verifying TSMID in the SD matches TSM certificate's TSM ID attribute

   * Now the TEE is ready to carry out update listed in the "content" message

2. Update action

   Commented [DT93]: If the request was valid, ...

   * If "newsdname" is given, replace the SD name for the SD to the new value

* If "newspid" is given, replace the SP ID assigned to this SD with the given new value

* If "spcert" is given, add this new SP certificate to the SD.

* If "deloldspcert" is present in the content, check previously assigned SP certificates to this SD, and delete the one that matches the given certificate hash value.

* If "renewteespaik" is given and has a value of~~as~~ "true", generate a new TEE SP AIK keypair, and replace the old one with this.

* Generate new DSI data if the request asks for updated DSI

* Now the TEE is ready to construct the response message

3. Construct UpdateSDResponse message

   * Create raw content

     + Operation status

     + "did" or full signer certificate information,

     + TEE SP AIK public key if DSI isn't going to be included

     + Updated DSI data if requested~~ if the request asks for it~~

   * The response message is encrypted with the same JWE CEK of the request without recreating a new content encryption key.

   * The encrypted message is signed with TEEpriv. The signer information ("did" or TEEcert) is encrypted.

4. Deliver response message. (a) The OTrP Agent returns this to the app; (b) The app passes this back to the TSM.

5. TSM processing. (a) The TSM processes the response message; (b) The TSM can look up the signer certificate from the device ID "did".

If a request is illegitimate or the signature doesn't pass, a "status" property in the response will indicate the error code and cause.

Commented [DT94]: (redundant)

8.2.2.3. UpdateSDResponse Message

The response message for a UpdateSDRequest contains the following content.

```
{
  "UpdateSDTBSResponse": {
    "ver": "1.0",
    "status": "<operation result>",
    "rid": "<the request ID received>",
    "tid": "<the transaction ID received>",
    "content": ENCRYPTED {
      "reason": "<failure reason detail>", // optional
      "did": "<the device id hash>",
      "cert": "<TEE certificate>", // optional
      "teespaik": "<TEE SP AIK public key, BASE64 encoded>",
      "teespaiktype": "<TEE SP AIK key type: RSA or ECC>",
      "dsi": "<Updated TEE state, including all SD owned by
        this TSM>"
    }
  }
}
```

In the response message, the following fields MUST be supplied.

did - The request should have known the signer certificate of this device from a prior request. This hash value of the device TEE certificate serves as a quick identifier only. A fFull device certificate isn't necessary.

teespaik - the newly generated SP AIK public key for the given SP if the TEE SP AIK for the SP is asked to be renewed in the request. This is an optional value if "dsi" is included in the response, which will contain all up--to--date TEE SP AIK key pairs.

Similar to the template for the creation of the encrypted and signed CreateSDResponse, the final UpdateSDResponse looks like the following.

```
{
  "UpdateSDResponse": {
    "payload": "<UpdateSDTBSResponse JSON above>",
    "protected": {
        "<BASE64URL of signing algorithm>"
    },
    "signature": "<signature contents signed by TEE device private
                  key (BASE64URL)>"
  }
}
```

A response message type "status" will be returned when the TEE ~~totally~~ fails to respond. The OTrP Agent is responsible to create this message.

```
{
  "status": {
     "result": "fail",
     "error-code": "ERR_TEE_UNKNOWN",
     "error-message": "TEE fails to respond"
  }
}
```

Commented [DT95]: Is this literal or just an example?

8.2.2.4. Error Conditions

An error may occur if a request isn't valid or the TEE runs into some error. The list of possible errors are the following. Refer to ~~section~~ the Error Code List (Section 14.1) for detailed causes and actions.

ERR_REQUEST_INVALID

ERR_UNSUPPORTED_MSG_VERSION

ERR_UNSUPPORTED_CRYPTO_ALG

ERR_DEV_STATE_MISMATCH

ERR_SD_NOT_FOUND

ERR_SDNAME_ALREADY_USED

ERR_SPCERT_INVALID

ERR_TEE_FAIL

ERR_TEE_UNKNOWN

ERR_TSM_NOT_AUTHORIZED

ERR_TSM_NOT_TRUSTED

8.2.3. DeleteSD

A TSM sends a DeleteSDRequest message to a TEE to delete a specified SD that it owns. An SD can be deleted only if there is no TA associated with this SD in the device. The request message can contain a flag to instruct the TEE to delete all related TAs in an SD and then delete the SD.

The target TEE will operate with the following logic.

1. Look_up the ~~given~~ SD specified in the request message

2. Check that the TSM owns the SD

3. Check that the device state hasn't changed since the last operation

4. Check whether there are TAs in this SD

5. If TA exists in an SD, check whether the request instructs whether the TA should be deleted. If the request instructs the TEE to delete TAs, delete all TAs in this SD. If the request doesn't instruct the TEE to delete TAs, return an error "ERR_SD_NOT_EMPTY".

6. Delete the SD

7. If this is the last SD of this SP, delete the TEE SP AIK key

8.2.3.1. DeleteSDRequest Message

The request message for DeleteSD has the following JSON format.

```
{
   "DeleteSDTBSRequest": {
     "ver": "1.0",
     "rid": "<unique request ID>",
     "tid": "<transaction ID>", // this may be from prior message
     "tee": "<TEE routing name from the DSI for the SD's target>",
     "nextdsi": "true | false",
     "dsihash": "<hash of DSI returned in the prior query>",
     "content": ENCRYPTED { // this piece of JSON will be encrypted
       "tsmid": "<TSMID associated with this SD>",
       "sdname": "<SD name for the domain to be updated>",
       "deleteta": "true | false"
     }
  }
}
```

In the message,

rid - A unique value to identify this request

tid - A unique value to identify this transaction. It can have the same value for the tid in the preceding GetDeviceStateRequest.

tee - TEE ID returned from the previous response GetDeviceStateResponse

nextdsi - Indicates whether the up-~~–~~to-~~–~~date Device State Information (DSI) ~~should~~ is to be returned in the response to this request.

dsihash - The BASE64 encoded SHA256 hash value of the DSI data returned in the prior TSM operation with this target TEE. This value is always included such that a receiving TEE can check whether the device state has changed since its last query. It helps enforce SD update order in the right sequence without accidently overwrite an update that was done simultaneously.

content - The "content" is a JSON encrypted message that includes actual input for the SD update. The standard JSON content encryption key (CEK) is used, and the CEK is encrypted by the target TEE's public key.

tsmid - SD owner claim by TSM - An SD owned by a TSM will be associated with a trusted identifier defined as an attribute in the signer TSM certificate.

sdname - the name of the target SD to be updated.

deleteta - the value should be 'true' or 'false'. If it is present and the value is 'true', the TEE should delete all TAs associated with the SD in the device.

Following the OTrP message template, the full request is signed message over the DeleteSDTBSRequest as follows.

```
{
    "DeleteSDRequest": {
        "payload": "<DeleteSDTBSRequest JSON above>",
        "protected": "<integrity-protected header contents>",
        "header": <non-integrity-protected header contents>,
        "signature": "<signature contents signed by TSM private key>"
    }
}
```

Commented [DT96]: No quotes?

TSM signer certificate is included in the "header" property.

8.2.3.2. Request processing requirements at a TEE

Upon receiving a request message DeleteSDRequest at a TEE, the TEE must validate a request:

1. Validate the JSON request message

    * Validate JSON message signing

    * Validate that the request TSM certificate is chained to a trusted CA that the TEE embeds as its trust anchor. The TSM certificate status check is generally not needed any more in this request. The prior request should have validated the TSM certificate's revocation status.

    * Compare dsihash with the TEE cached last response DSI data to this TSM

    * Decrypt to get the plaintext of the content

    * Check that the target SD name is supplied

    * Check whether the requested SD exists

    * Check that the TSM owns this TSM by verifying that the TSMID in the SD matches the TSM certificate's TSM ID attribute

    * Now the TEE is ready to carry out the update listed in the "content" message

2. Deletion action

   * Check TA existence in this SD

   * If "deleteta" is "true", delete all TAs in this SD. If the value of "deleteta" is "false" and some TA exists, return an error "ERR_SD_NOT_EMPTY"

   * Delete the SD

   * Delete the TEE SP AIK key pair if this SD is the last one for the SP

   * Now the TEE is ready to construct the response message

3. Construct DeleteSDResponse message

   * Create response content

     + Operation status

     + "did" or full signer certificate information,

     + Updated DSI data if requested ~~if the request asks for it~~

   * The response message is encrypted with the same JWE CEK of the request without recreating a new content encryption key.

   * The encrypted message is signed with TEEpriv. The signer information ("did" or TEEcert) is encrypted.

4. Deliver response message. (a) The OTrP Agent returns this to the app; (b) The app passes this back to the TSM.

5. TSM processing. (a) The TSM processes the response message; (b) The TSM can look up signer certificate from the device ID "did".

If a request is illegitimate or signature doesn't pass, a "status" property in the response will indicate the error code and cause.

8.2.3.3. DeleteSDResponse Message

The response message for a DeleteSDRequest contains the following content.

```
{
  "DeleteSDTBSResponse": {
    "ver": "1.0",
    "status": "<operation result>",
    "rid": "<the request ID received>",
    "tid": "<the transaction ID received>",
    "content": ENCRYPTED {
      "reason": "<failure reason detail>", // optional
      "did": "<the device id hash>",
      "dsi": "<Updated TEE state, including all SD owned by
        this TSM>"
    }
  }
}
```

In the response message, the following fields MUST be supplied.

did - The request should have known the signer certificate of this device from a prior request. This hash value of the device TEE certificate serves as a quick identifier only. A fFull device certificate isn't necessary.

The final DeleteSDResponse looks like the following.

```
{
  "DeleteSDResponse": {
    "payload": "<DeleteSDTBSResponse JSON above>",
    "protected": {
        "<BASE64URL of signing algorithm>"
    },
    "signature": "<signature contents signed by TEE device
      private key (BASE64URL)>"
  }
}
```

Commented [DT97]: If it's JSON, why does it have to be in quotes here?

A response message type "status" will be returned when TEE totally fails to respond. OTrP Agent is responsible to create this message.

```
{
  "status": {
     "result": "fail",
     "error-code": "ERR_TEE_UNKNOWN",
     "error-message": "TEE fails to respond"
  }
}
```

8.2.3.4. Error Conditions

An error may occur if a request isn't valid or the TEE runs into some error. The list of possible errors ~~are the following~~is as follows. Refer to ~~section~~ the Error Code List (Section 14.1) for detailed causes and actions.

ERR_REQUEST_INVALID

ERR_UNSUPPORTED_MSG_VERSION

ERR_UNSUPPORTED_CRYPTO_ALG

ERR_DEV_STATE_MISMATCH

ERR_SD_NOT_EMPTY

ERR_SD_NOT_FOUND

ERR_TEE_FAIL

ERR_TEE_UNKNOWN

ERR_TSM_NOT_AUTHORIZED

ERR_TSM_NOT_TRUSTED

8.3. Trusted Application Management

This protocol doesn't introduce a TA container concept. All ~~the~~ TA authorization and management will be up to the TEE implementation.

The following three TA management commands ~~will be~~are supported.

o InstallTA - provision a TA by TSM

o UpdateTA - update a TA by TSM

o DeleteTA - remove TA registration information with a SD, remove the TA binary from TEE, and remove all TA-~~-~~related data in a TEE

8.3.1. InstallTA

TA binary data can be from two sources:

1. A TSM supplies the signed TA binary

2. A Client Application supplies the TA binary

Commented [DT98]: This discussion should be in the document introduction.

This specification considers only the first case where a TSM supplies a TA binary. When such a request is received by a TEE, an SD is already created and is ready ~~to take~~for TA installation.

Commented [DT99]: Why? This seems like an important BOF/WG scoping discussion to have.

A TSM sends the following information in a ~~message~~ InstallTARequest message to a target TEE:

o The target SD information: SP ID and SD name

o Encrypted TA binary data. TA data is encrypted with the TEE SP AIK.

o TA metadata. It is optional to include the SP signer certificate for the SD to add if the SP has changed signer since the SD was created.

The TEE processes command given by the TSM to install a TA into an SP's SD. It does the following:

o Validation

   * The TEE validates the TSM message authenticity

   * Decrypt to get request content

   * Look up the SD with the SD name

   * Checks that the TSM owns the SD

   * Checks that the DSI hash matches that the device state hasn't changed

Commented [DT100]: Fix grammar

o TA validation

   * Decrypt to get TA binary and any personalization data with "TEE SP AIK private key"

   * Check that SP ID is the one that is registered with the SP SD

   * Check that the TA signer is either the newly given SP certificate or the one in the SD. The TA signing method is specific to the TEE. This specification doesn't define how a TA should be signed.

Commented [DT101]: Is the TSM expected to know it somehow?

   * If a TA signer is given in the request, add this signer into the SD.

o TA installation

   * The TEE re-encrypts the TA binary and its personalization data with its own method

   * The ~~onto~~ in ~~TEE~~ secure storage ~~area~~.

o Construct a response message. This involves signing ~~a~~ encrypted status information for the requesting TSM.

8.3.1.1. InstallTARequest Message

The request message for InstallTA has the following JSON format.


```
{
  "InstallTATBSRequest": {
    "ver": "1.0",
    "rid": "<unique request ID>",
    "tid": "<transaction ID>",
    "tee": "<TEE routing name from the DSI for the SD's target>",
    "nextdsi": "true | false",
    "dsihash": "<hash of DSI returned in the prior query>",
    "content": ENCRYPTED {
      "tsmid": "<TSM ID previously assigned to the SD>",
      "spid": "<SPID value>",
      "sdname": "<SD name for the domain to install the TA>",
      "spcert": "<BASE64 encoded SP certificate >", // optional
      "taid": "<TA identifier>"
    },
    "encrypted_ta": {
      "key": "<A 256-bit symmetric key encrypted by TEEspaik public
              key>",
      "iv": "<hex of 16 random bytes>",
      "alg": "<encryption algoritm. AESCBC by default.",
      "ciphertadata": "<BASE64 encoded encrypted TA binary data>",
      "cipherpdata": "<BASE64 encoded encrypted TA personalization
                      data>"
    }
  }
}
```


In the message,

rid - A unique value to identify this request

tid - A unique value to identify this transaction. It can have the same value for the tid in the preceding GetDeviceStateRequest.

tee - TEE ID returned from the previous ~~response~~ GetDeviceStateResponse

nextdsi - Indicates whether the up-~~-~~to-~~-~~date Device State Information (DSI) ~~should~~ is to be returned in the response to this request.

dsihash - The BASE64-~~-~~encoded SHA256 hash value of the DSI data returned in the prior TSM operation with this target TEE. This value is always included such that a receiving TEE can check whether the device state has changed since its last query. It helps enforce SD update order in the right sequence without accidently overwrite an update that was done simultaneously.

content - The "content" is a JSON encrypted message that includes actual input for the SD update. The standard JSON content encryption key (CEK) is used, and the CEK is encrypted by the target TEE's public key.

tsmid - SD owner claim by TSM - An SD owned by a TSM will be associated with a trusted identifier defined as an attribute in the signer TSM certificate.

spid - SP identifier of the TA owner SP

sdname - the name of the target SD where the TA ~~should~~ is to be installed

spcert - an optional field to specify the SP certificate that signed the TA. This is sent if the SP has a new certificate that hasn't been previously registered with the target SD where the TA should be installed.

taid - the identifier of the TA application to be installed

encrypted_ta - the message portion contains encrypted TA binary data and personalization data. The TA data encryption key is placed in "key", which is encrypted by the recipient's public key. The TA data encryption uses symmetric key based encryption such as AESCBC.

Commented [DT102]: What are the requirements?

Following the OTrP message template, the full request is a signed message over the InstallTATBSRequest as follows.

```
{
    "InstallTARequest": {
        "payload": "<InstallTATBSRequest JSON above>",
        "protected": "<integrity-protected header contents>",
        "header": <non-integrity-protected header contents>,
        "signature": "<signature contents signed by TSM private key>"
    }
}
```

Commented [DT103]: Why does payload put a JSON payload in quotes, and header is not a string? This looks backwards.

8.3.1.2. InstallTAResponse Message

The response message for a InstallTARequest contains the following content.

```
{
  "InstallTATBSResponse": {
    "ver": "1.0",
    "status": "<operation result>",
    "rid": "<the request ID received>",
    "tid": "<the transaction ID received>",
    "content": ENCRYPTED {
      "reason": "<failure reason detail>", // optional
      "did": "<the device id hash>",
      "dsi": "<Updated TEE state, including all SD owned by
        this TSM>"
    }
  }
}
```

In the response message, the following fields MUST be supplied.

did - the SHA256 hash of the device TEE certificate. This shows the device ID explicitly to the receiving TSM.

The final message InstallTAResponse looks like the following.

```
{
    "InstallTAResponse": {
        "payload": "<InstallTATBSResponse JSON above>",
        "protected": {
            "<BASE64URL of signing algorithm>"
        },
        "signature": "<signature contents signed by TEE device
          private key (BASE64URL)>"
    }
}
```

A response message type "status" will be returned when the TEE ~~totally~~ fails to respond. The OTrP Agent is responsible to create this message.

```
{
  "status": {
     "result": "fail",
     "error-code": "ERR_TEE_UNKNOWN",
     "error-message": "TEE fails to respond"
  }
}
```


8.3.1.3. Error Conditions

An error may occur if a request isn't valid or the TEE runs into some error. The list of possible errors are ~~the following~~as follows. Refer to ~~section~~ the Error Code List (Section 14.1) for detailed causes and actions.

ERR_REQUEST_INVALID

ERR_UNSUPPORTED_MSG_VERSION

ERR_UNSUPPORTED_CRYPTO_ALG

ERR_DEV_STATE_MISMATCH

ERR_SD_NOT_FOUND

ERR_TA_INVALID

ERR_TA_ALREADY_INSTALLED

ERR_TEE_FAIL

ERR_TEE_UNKNOWN

ERR_TEE_RESOURCE_FULL

ERR_TSM_NOT_AUTHORIZED

ERR_TSM_NOT_TRUSTED

8.3.2. UpdateTA

This TSM-~~-~~initiated command can update a TA and its data in an SP's SD that it manages for the following purposes.

1. Update TA binary

2. Update TA's personalization data

The TSM presents the proof of the SD ownership to a TEE, and includes related information in its signed message. The entire request is also encrypted for ~~the~~ end-to-end confidentiality.

The TEE processes the command ~~given~~ from ~~by~~ the TSM to update the TA of an SP SD. It does the following:

o Validation

  * The TEE validates the TSM message authenticity

  * Decrypt to get request content

  * Look up the SD with the SD name

  * Checks that the TSM owns the SD

  * Checks that the DSI hash matches that the device state hasn't changed

Commented [DT104]: Fix grammar

o TA validation

  * Both TA binary and personalization data are optional, but at least one of them shall be present in the message

  * Decrypt to get the TA binary and any personalization data with "TEE SP AIK private key"

  * Check that the SP ID is the one that is registered with the SP SD

  * Check that the TA signer is either the newly given SP certificate or the one in SD. The TA signing method is specific to TEE. This specification doesn't define how a TA should be signed.

  * If a TA signer is given in the request, add this signer into the SD.

o TA update

  * The TEE re-encrypts the TA binary and its personalization data with its own method

  * The TEE replaces the existing TA binary and its personalization data with the new binary and data.

o Construct a response message. This involves signing ~~a~~ encrypted status information for the requesting TSM.

8.3.2.1. UpdateTARequest Message

The request message for UpdateTA has the following JSON format.

```
{
  "UpdateTATBSRequest": {
    "ver": "1.0",
    "rid": "<unique request ID>",
    "tid": "<transaction ID>",
    "tee": "<TEE routing name from the DSI for the SD's target>",
    "nextdsi": "true | false",
    "dsihash": "<hash of DSI returned in the prior query>",
    "content": ENCRYPTED {
      "tsmid": "<TSM ID previously assigned to the SD>",
      "spid": "<SPID value>",
      "sdname": "<SD name for the domain to be created>",
      "spcert": "<BASE64 encoded SP certificate >", // optional
      "taid": "<TA identifier>"
    },
    "encrypted_ta": {
      "key": "<A 256-bit symmetric key encrypted by TEEspaik public
              key>",
      "iv": "<hex of 16 random bytes>",
      "alg": "<encryption algorithm. AESCBC by default.",
      "ciphernewtadata": "<Change existing TA binary to this new TA
          binary data(BASE64 encoded and encrypted)>",
      "ciphernewpdata": "<Change the existing data to this new TA
          personalization data(BASE64 encoded and encrypted)>"
          // optional
    }
  }
}
```

In the message,

rid - A unique value to identify this request

tid - A unique value to identify this transaction. It can have the same value for the tid in the preceding GetDeviceStateRequest.

tee - TEE ID returned from the previous ~~response~~ GetDeviceStateResponse.

nextdsi - Indicates whether the up-~~-~~to-~~-~~date Device State Information (DSI) ~~should ~~is to be returned in the response to this request.

dsihash - The BASE64-~~-~~encoded SHA256 hash value of the DSI data returned in the prior TSM operation with this target TEE. This value is always included such that a receiving TEE can check whether the device state has changed since its last query. It helps enforce SD update order in the right sequence without accidently overwrite an update that was done simultaneously.

content - The "content" is a JSON encrypted message that includes actual input for the SD update. The standard JSON content encryption key (CEK) is used, and the CEK is encrypted by the target TEE's public key.

tsmid - SD owner claim by TSM - An SD owned by a TSM will be associated with a trusted identifier defined as an attribute in the signer TSM certificate.

spid - SP identifier of the TA owner SP

spcert - an optional field to specify the SP certificate that signed the TA. This is sent if the SP has a new certificate that hasn't been previously registered with the target SD where the TA ~~should~~ is to be installed.

sdname - the name of the target SD where the TA ~~should~~ is to be updated

taid - an identifier for the TA application to be updated

encrypted_ta - the message portion contains new encrypted TA binary data and personalization data.

Following the OTrP message template, the full request is signed message over the UpdateTATBSRequest as follows.

```
{
    "UpdateTARequest": {
        "payload": "<UpdateTATBSRequest JSON above>",
        "protected": "<integrity-protected header contents>",
        "header": <non-integrity-protected header contents>,
        "signature": "<signature contents signed by TSM private key>"
    }
}
```

8.3.2.2. UpdateTAResponse Message

The response message for a UpdateTARequest contains the following content.

```
{
  "UpdateTATBSResponse": {
    "ver": "1.0",
    "status": "<operation result>",
    "rid": "<the request ID received>",
    "tid": "<the transaction ID received>",
    "content": ENCRYPTED {
      "reason": "<failure reason detail>", // optional
      "did": "<the device id hash>",
      "dsi": "<Updated TEE state, including all SD owned by
        this TSM>"
    }
  }
}
```

In the response message, the following fields MUST be supplied.

did - the SHA256 hash of the device TEE certificate. This shows the device ID explicitly to the receiving TSM.

The final message UpdateTAResponse looks like the following.

```
{
    "UpdateTAResponse": {
        "payload":"<UpdateTATBSResponse JSON above>",
        "protected": {
            "<BASE64URL of signing algorithm>"
        },
        "signature": "<signature contents signed by TEE device
          private key (BASE64URL)>"
    }
}
```

A response message type "status" will be returned when the TEE ~~totally~~ fails to respond. The OTrP Agent is responsible to create this message.

```
{
  "status": {
     "result": "fail",
     "error-code": "ERR_TEE_UNKNOWN",
     "error-message": "TEE fails to respond"
  }
}
```

8.3.2.3. Error Conditions

An error may occur if a request isn't valid or the TEE runs into some error. The list of possible errors are ~~the following~~as follows. Refer to ~~section~~ the Error Code List (Section 14.1) for detailed causes and actions.

ERR_REQUEST_INVALID

ERR_UNSUPPORTED_MSG_VERSION

ERR_UNSUPPORTED_CRYPTO_ALG

ERR_DEV_STATE_MISMATCH

ERR_SD_NOT_FOUND

ERR_TA_INVALID

ERR_TA_NOT_FOUND

ERR_TEE_FAIL

ERR_TEE_UNKNOWN

ERR_TSM_NOT_AUTHORIZED

ERR_TSM_NOT_TRUSTED

8.3.3. DeleteTA

This operation defines OTrP messages that allow a TSM to instruct a TEE to delete a TA for an SP in a given SD. A TEE will delete a TA from an SD and also TA data in the TEE. A Client Application cannot directly access a TEE or OTrP Agent to delete a TA.

8.3.3.1. DeleteTARequest Message

The request message for DeleteTA has the following JSON format.

```
{
  "DeleteTATBSRequest": {
    "ver": "1.0",
    "rid": "<unique request ID>",
    "tid": "<transaction ID>",
    "tee": "<TEE routing name from the DSI for the SD's target>",
    "nextdsi": "true | false",
    "dsihash": "<hash of DSI returned in the prior query>",
    "content": ENCRYPTED {
      "tsmid": "<TSM ID previously assigned to the SD>",
      "sdname": "<SD name of the TA>",
      "taid": "<the identifier of the TA to be deleted from the
               specified SD>"
    }
  }
}
```

In the message,

rid - A unique value to identify this request

tid - A unique value to identify this transaction. It can have the same value for the tid in the preceding GetDeviceStateRequest.

tee - The TEE ID returned from the previous ~~response~~ GetDeviceStateResponse.

nextdsi - Indicates whether the up-~~-~~to-~~-~~date Device State Information (DSI) ~~should~~ is to be returned in the response to this request.

dsihash - The BASE64-~~-~~encoded SHA256 hash value of the DSI data returned in the prior TSM operation with this target TEE. This value is always included such that a receiving TEE can check whether the device state has changed since its last query. It helps enforce SD update order in the right sequence without accidently overwrite an update that was done simultaneously.

content - The "content" is a JSON encrypted message that includes actual input for the SD update. The standard JSON content encryption key (CEK) is used, and the CEK is encrypted by the target TEE's public key.

tsmid - SD owner claim by TSM - An SD owned by a TSM will be associated with a trusted identifier defined as an attribute in the signer TSM certificate.

sdname - the name of the target SD where the TA is installed

taid - an identifier for the TA application to be deleted

Following the OTrP message template, the full request is a signed message over the DeleteTATBSRequest as follows.

```
{
    "DeleteTARequest": {
        "payload": "<DeleteTATBSRequest JSON above>",
        "protected": "<integrity-protected header contents>",
        "header": -<non-integrity-protected header contents>,
        "signature":"<signature contents signed by TSM
            private key>"
    }
}
```

8.3.3.2. Request processing requirements at a TEE

A TEE processes a command given by a TSM to delete a TA of an SP SD. It does the following:

1. Validate the JSON request message

    * The TEE validates TSM message authenticity

    * Decrypt to get request content

    * Look up the SD and the TA with the given SD name and TA ID

    * Checks that the TSM owns the SD, and TA is installed in the SD

    * Checks DSI hash matches that the device state hasn't changed

2. Deletion action

    * If all the above validation points pass, the TEE deletes the TA from the SD

    * The TEE may also delete all personalization data for the TA

3. Construct DeleteTAResponse message.

If a request is illegitimate or the signature doesn't pass, a "status" property in the response will indicate the error code and cause.

Commented [DT105]: MAY? SHOULD? MUST?

8.3.3.3. DeleteTAResponse Message

The response message for a DeleteTARequest contains the following content.

```
{
  "DeleteTATBSResponse": {
    "ver": "1.0",
    "status": "<operation result>",
    "rid": "<the request ID received>",
    "tid": "<the transaction ID received>",
    "content": ENCRYPTED {
      "reason": "<failure reason detail>", // optional
      "did": "<the device id hash>",
      "dsi": "<Updated TEE state, including all SD owned by
        this TSM>"
    }
  }
}
```

In the response message, the following fields MUST be supplied.

did - the SHA256 hash of the device TEE certificate. This shows the device ID explicitly to the receiving TSM.

The final message DeleteTAResponse looks like the following.

```
{
    "DeleteTAResponse": {
        "payload": "<DeleteTATBSResponse JSON above>",
        "protected": {
            "<BASE64URL of signing algorithm>"
        },
        "signature": "<signature contents signed by TEE device
            private key (BASE64URL)>"
    }
}
```

A response message type "status" will be returned when the TEE ~~totally~~ fails to respond. The OTrP Agent is responsible to create this message.

```
{
  "status": {
     "result": "fail",
     "error-code": "ERR_TEE_UNKNOWN",
     "error-message": "TEE fails to respond"
  }
}
```


8.3.3.4. Error Conditions

An error may occur if a request isn't valid or the TEE runs into some error. The list of possible errors are the followingas follows. Refer to section the Error Code List (Section 14.1) for detailed causes and actions.

ERR_REQUEST_INVALID

ERR_UNSUPPORTED_MSG_VERSION

ERR_UNSUPPORTED_CRYPTO_ALG

ERR_DEV_STATE_MISMATCH

ERR_SD_NOT_FOUND

ERR_TA_NOT_FOUND

ERR_TEE_FAIL

ERR_TEE_UNKNOWN

ERR_TSM_NOT_AUTHORIZED

ERR_TSM_NOT_TRUSTED

9. Response Messages a TSM May Expect

A TSM expects some feedback from a remote device when a request message is delivered to a device. The following three types of responses SHOULD be supplied.

Type 1: Expect a valid TEE--generated response message

A valid TEE signed response may contain errors detected by TEE, e.g., tje TSM is trusted but TSM--supplied data is missing, for example, SP ID doesn't exist. TEE MUST be able to sign and encrypt.

If a TEE isn't able to sign a response, the TEE ~~should~~ returns an error to the OTrP Agent without giving any other internal information. The OTrP Agent will ~~be~~ generate~~ing~~ the response.

Type 2: OTrP Agent generated error message when TEE fails. OTrP Agent errors will be defined in this document.

A Type 2 message has the following format.

```
{
  "OTrPAgentError": {
      "ver": "1.0",
      "rid": "",
      "tid": "",
      "errcode": "ERR_TEE_FAIL | ERR_TEE_BUSY"
  }
}
```

Type 3: The OTrP Agent itself isn't reachable or fails. A Client Application is responsible to handle error and response TSM in its own way. This is out of scope for this specification.

10. Basic Protocol Profile

This section describes a baseline for interoperability among the protocol entities, mainly, the TSM and TEE.

A TEE MUST support RSA algorithms. It is optional to support ECC algorithms. A TSM should use a RSA certificate for TSM message signing. It may use an ECC certificate if it detects that the TEE supports ECC.

A TSM MUST support both RSA 2048-bit algorithm and ECC P-256 algorithms. With this, a TEE and TFW certificate can be either RSA or ECC type.

JSON signing algorithms

o RSA PKCS#1 with SHA256 signing : "RS256"

o ECDSA with SHA256 signing : "ES256"

JSON asymmetric encryption algorithms (describes key-exchange or key-agreement algorithm for sharing symmetric key with TEE):

o RSA PKCS#1 : "RSA1_5"

Commented [DT106]: Can't parse this phrase

Commented [DT107]: MAY?

Commented [DT108]: SHOULD?

Commented [DT109]: Mention what field in the protocol it uses to know this.

o ECDH using TEE ECC P-256 key and ephemeral ECC key generated by TSM : "ECDH-ES+A128W"

JSON symmetric encryption algorithms (describes symmetric algorithm for encrypting body of data, using symmetric key transferred to TEE using asymmetric encryption):

o Authenticated encryption AES 128 CBC with SHA256 : {"enc":"A128CBC-HS256"}

11. Attestation Implementation Consideration

It is important to know that the state of a device is appropriate before trusting that a device is what it says it is. The attestation scheme for OTrP must also be able to cope with different TEEs, including those that are OTrP--compliant and those that use another mechanism. In the initial version, only one active TEE is assumed.

It is out of scope about how the TSM and the device implement the trust hierarchy verification. However, it is helpful to understand what each system provider should do in order to properly implement an OTrP trust hierarchy.

In this section, we provide some implementation reference consideration.

11.1. OTrP Secure Boot Module

11.1.1. Attestation signer

It is proposed that attestation for OTrP is based on the SBM secure boot layer, and that further attestation is not performed within the TEE itself during security domain operations. The rationale is that the device boot process will be defined to start with a secure boot approach that, using eFuse, only releases attestation signing capabilities into the SBM once a secure boot has been established. In this way, the release of the attestation signer can be considered the first "platform configuration metric", using TCG terminology.

Commented [DT110]: Expand acronym on first use

11.1.2. SBM initial requirements

R1 The SBM must be possible to load securely into the secure boot flow

R2 The SBM must allow a public / private key pair to be generated during device manufacture

R3 The public key and certificate must be possible to store securely ~~from tamper~~

R4 The private key must be possible to store encrypted at rest

R5 The private key must only be visible to the SBM when it is decrypted

R6 The SBM must be able to read a list of root and intermediate certificates that it can use to check certificate chains with. The list must be stored such that it cannot be tampered with

R7 Possible need to allow a TEE to access its unique TEE specific private key

Commented [DT111]: Not sure I understand the "possible need", this seems like a hard requirement to me.

11.2. TEE Loading

During boot, the SBM is required to start all of the ROOT TEEs. Before loading them, the SBM must first determine whether the code sign signature of the TEE is valid. If TEE integrity is confirmed, ~~it~~ the TEE may be started. The SBM must then be able to receive the identity certificate from the TEE (if that TEE is OTrP compliant). The identity certificate and keys will need to be baked into the TEE image, and therefore also covered by the code signer hash during the manufacturing~~e~~ process. The private key for the identity certificate must be securely protected. The private key for a TEE identity must never be released no matter how the public key and certificate are released to the SBM.

Commented [DT112]: Undefined term

Commented [DT113]: MAY?

Once the SBM has successfully booted a TEE and retrieved the identity Certificate, ~~it~~ the SBM will commit this to the platform configuration register (PCR) set~~,~~ for later use during attestation. At~~s~~ ~~a~~ minimum, the following data must be committed to the PCR for each TEE:

1. Public key and certificate for the TEE

2. TEE reference that can be used later by a TSM to identify this TEE

Commented [DT114]: "identifier"?

11.3. Attestation Hierarchy

The attestation hierarchy and seed required for TSM protocol operation must be built into the device at manufacture. Additional TEEs can be added post~~-~~-manufacture using the scheme proposed, ~~however~~but it is outside of the current scope of this document to detail that.

It should be noted that the attestation scheme described is based on signatures. The only encryption that takes place is with eFuse to release the SBM signing key and later during the protocol lifecycle management interchange with the TSM.

11.3.1. Attestation hierarchy establishment: manufacture

During manufacture the following steps are required:

1. A d~~D~~evice-~~-~~specific TFW key pair and certificate are burnt into the device, encrypted by eFuse. This key pair will be used for signing operations performed by the SBM.

2. TEE images are loaded and include a TEE instance-~~-~~specific key pair and certificate. The key pair and certificate are included in the image and covered by the code signing hash.

3. The process for TEE images is repeated for any subordinate TEEs

Commented [DT115]: Undefined term

11.3.2. Attestation hierarchy establishment: device boot

During device boot the following steps are required:

1. Secure boot releases the TFW private key by decrypting with eFuse

Commented [DT116]: To the TFW?

2. The SBM verifies the code-signing signature of the active TEE and places its TEE public key into a signing buffer, along with their reference for later access. For a non-OTrP TEE, the SBM leaves the TEE public key field blank.

Commented [DT117]: “its identifier”?

3. The SBM signs the signing buffer with the TFW private key.

4. Each active TEE performs the same operation as the SBM, building up their own signed buffer containing subordinate TEE information.

11.3.3. Attestation hierarchy establishment: TSM

Before a TSM can begin operation in the marketplace, it must obtain a TSM key pair and certificate (TSMpub, TSMpriv) from a CA that is registered in the trust store of the TEE. In this way, the TEE can check the intermediate and root CA and verify that it trusts this TSM to perform operations on the TEE.

Commented [DT118]: This needs to be made clear in the trust model up front.

12. Acknowledgements

We thank Alin Mutu for his contribution to many discussion that helped to design the trust flow mechanisms, and the creation of the flow diagrams. We also thank the following people (~~by~~ in alphabetical order) for their input and review: Sangsu Baek, Marc Canel, Roger Casals, Rob Coombs, Lubna Dajani, Richard Parris, and Pengfei Zhao.

13. Contributors

Brian Witten
Symantec
900 Corporate Pointe
Culver City, CA 90230
USA

Email: brian_witten@symantec.com

Tyler Kim
Solacia
5F, Daerung Post Tower 2, 306 Digital-ro
Seoul 152-790
Korea

Email: tkkim@sola-cia.com

14. IANA Considerations

The error code listed in the next section will be registered.

14.1. Error Code List

This section lists error codes that could be reported by a TA or TEE in a device in responding to a TSM request.

ERR_DEV_STATE_MISMATCH - A TEE will return this error code if the DSI hash value from TSM doesn't match ~~with that~~ the hash value of the device's current DSI.

ERR_SD_ALREADY_EXIST - This error will occur if an SD to be created already exists in the TEE.

ERR_SD_NOT_EMPTY - This is reported if a target SD isn't empty.

ERR_SDNAME_ALREADY_USED ~~-~~ A TEE will return this error code if the new SD name already exists in the namespace of the TSM in the TEE.

ERR_REQUEST_INVALID - This error will occur if the TEE meets any of the following conditions with a request message: (1) The request from a TSM has an invalid message structure; mandatory information is absent in the message. undefined member or structure is included. (2) TEE fails to verify signature of the message or fails to decrypt its contents. ~~(3) etc.~~

ERR_SPCERT_INVALID - If a new SP certificate for the SD to be updated is not valid, then the TEE will return this error code.

Commented [DT119]: This section does not follow RFC 5226

Commented [DT120]: Not true. ERR_TEE_FAIL says it's reported by an OTrP Agent not a TA or a TEE.

I think this section should be split into three separate lists.
1)Errors returned by a TA
2)Errors returned by a TEE
3)Errors returned by an OTrP Agent.

Commented [DT121]: Grammar: EXISTS

Commented [DT122]: In the SD?

ERR_TA_ALREADY_INSTALLED - W~~w~~hile installing a TA, a TEE will return this error if the TA already has been installed in the SD.

ERR_TA_INVALID - This error will occur when a TEE meets any of following conditions while checking the validity of a TA: (1) The TA binary has a format that the TEE can't recognize. (2) The TEE fails to decrypt the encoding of the TA binary and personalization data. (3) If an SP isn't registered with the SP SD where the TA will be installed.~~ (4) etc.~~

ERR_TA_NOT_FOUND - This error will occur~~s~~ when the target TA doesn't exist in the SD.

ERR_TEE_BUSY - The device TEE is busy. The request should be generally sent later to retry.

ERR_TEE_FAIL - The TEE fail~~s~~eds to respond to a TSM request. The OTrP Agent will construct an error message in respon~~ding~~se to the TSM's request. ~~And a~~Also if the TEE fails to process a request because of ~~its~~an internal error, it will return this error code.

ERR_TEE_RESOURCE_FULL - This error is reported when a device resource isn't available anymore such as storage space is full.

ERR_TEE_UNKNOWN - This error will occur if the receiver TEE is not supposed to receive the request. That will be determined by checking the TEE name or device id in the request message.

ERR_TFW_NOT_TRUSTED - A TEE ~~may concern~~is responsible for determining that the underlying device firmware is trustworthy. If the TEE determines the TFW is not trustworthy, then this error will occur.

ERR_TSM_NOT_TRUSTED - Before processing a request, a TEE needs to make sure whether the sender TSM is trustworthy by checking the validity of the TSM certificate, etc. If the TEE finds that the TSM is not ~~reliable~~trustworthy, then it will return this error code.

ERR_UNSUPPORTED_CRYPTO_ALG - This error will occur if a TEE receives a request message encoded with cryptographic algorithms that the TEE doesn't support.

ERR_UNSUPPORTED_MSG_VERSION - This error will occur if a TEE receives ~~the~~ a message version ~~of message~~ that the TEE can't deal with.

15. Security Consideration

Commented [DT123]: "can"

Commented [DT124]: How much later?

Commented [DT125]: As noted earlier, there should be separate errors for the TEE returning an internal error vs an OTrP Agent claiming that a TEE didn't respond.

15.1. Cryptographic Strength

The strength of the cryptographic algorithms, using the measure of 'bits of security' defined in NIST SP800-57, allowed for the OTrP protocol is:

- At a minimum, 112 bits of security. The limiting factor for this is the RSA-2048 algorithm, which is indicated as providing 112 bits of symmetric key strength in SP800-57. It is important that RSA is supported in order to enhance the interoperability of the protocol.

- The option exists to choose algorithms providing 128 bits of security. This requires using TEE devices that support ECC P256.

The available algorithms and key sizes specified in this document are based on industry standards. Over time the recommended or allowed cryptographic algorithms may change. It is important that the OTrP protocol allows for crypto-agility.

Commented [DT126]: ("OTrP protocol" is redundant since the P stands for protocol)

Commented [DT127]: How is the crypto algorithm negotiated between a TEE and a TSM? Reference a section that specifies the algorithm negotiation procedure.

15.2. Message Security

OTrP messages between the TSM and TEE are protected by message security using JWS and JWE. The 'Basic protocol profile' section of this document describes the algorithms used for this. All OTrP TEE devices and OTrP TSMs must meet the requirements of the basic profile. In the future additional 'profiles' can be added.

PKI is used to ensure that the TEE will only communicate with a trusted TSM, and to ensure that the TSM will only communicate with a trusted TEE.

15.3. TEE Attestation

It is important that the TSM can trust that it is talking to a trusted TEE. This is achieved through attestation. The TEE has a private key and certificate built into it at manufacture, which is used to sign data supplied by the TSM. This allows the TSM to verify that the TEE is trusted.

It is also important that the TFW (trusted firmware) can be checked. The TFW has a private key and certificate built into it at manufacturer, which allows the TEE to check that that the TFW is trusted.

The GetDeviceState message therefore allows the TSM to check that it trusts the TEE, and the TEE at this point will check whether it trusts the TFW.

15.4. TA Protection

A TA will be delivered in an encrypted form. This encryption is an additional layer within the message encryption described in the 'Basic protocol profile' section of this document. The TA binary is encrypted for each target device with the device's TEE SP AIK public key. A TSM ~~may~~ can either do this encryption itself or provide~~s~~ the TEE SP AIK public key to a SP such that the SP encrypts the encrypted TA ~~to TSM~~ for distribution to the TEE.

Commented [DT128]: Give section number

The encryption algorithm can use a random~~ly~~ AES 256 key "taek" with a 16 byte random IV, and the "taek" is encrypted by the "TEE SP AIK public key". The following encrypted TA data structure is expected by a TEE:

```
"encrypted_ta_bin": {
  "key": "<A 256-bit symmetric key encrypted by TEE SP AIK public
           key>",
  "iv": <hex of 16 random bytes>",
  "alg": "AESCBC",
  "cipherdata": "<BASE64 encoded encrypted TA binary data>"
}
```

15.5. TA Personalization Data

An SP or TSM can supply personalization data for a TA to initialize for a device. Such data is passed through an InstallTA command from a TSM. The personalization data itself is (or can be) opaque to the TSM. The data can be from the SP without being revealed to the TSM. The data is sent in an encrypted manner in a request to a device such that only the device can decrypt. A device's TEE SP AIK public key for an SP is used to encrypt the data.

```
"encrypted_ta_data": { // "TA personalization data"
    "key": "<A 256-bit symmetric key encrypted by TEE SP AIK public
            key>",
    "iv": "<hex of 16 random bytes>",
    "alg": "AESCBC",
    "cipherdata": "<BASE64-~~-~~encoded encrypted TA personalization
                   data>"
  }
```

15.6. TA trust check at TEE

A TA binary is signed by a TA signer certificate. This TA signing certificate/private key belongs to the SP, and may be self-signed (i.e., ~~–~~ it need not participate in a trust hierarchy). It is the responsibility of the TSM to only allow verified TAs from trusted SPs into the system. Delivery of that TA to the TEE is then the responsibility of the TEE, using the security mechanisms provided by the OTrP protocol.

We allow a way for an (untrusted) application to check the trustworthinessy of a TA. An OTrP Agent will have a function to allow an application to query the metadata of a TA.

Commented [DT129]: MUST?

Commented [DT130]: What metadata? What's the message format?

An application in the Rich O/S may perform verification of the TA by verifying the signature of the TA. The OTRPService.getTAInformation() function is available to return the TEE-supplied TA signer and TSM signer information to the untrusted application. An application can do additional trust checks on the certificate returned for this TA. It may might trust the TSM, or require additional SP signer trust chaining.

15.7. One TA Multiple SPs Case

A TA for different multiple SPs must have a different identifier per SP. A TA will be installed in a different SD for the each respective SP.

15.8. OTrP Agent Trust Model

An OTrP Agent could be malware in the vulnerable Android Rich OS. A Client Application will connect its TSM provider for required TA installation. It gets command messages from the TSM, and passes the message to the OTrP Agent.

Commented [DT131]: IETF docs should never be OS specific.

The OTrP protocol is a conduit for enabling the TSM to communicate with the device's TEE to manage SDs and TAs. All TSM messages are signed and sensitive data is encrypted such that the OTrP Agent cannot modify or capture sensitive data.

15.9. OCSP Stapling Data for TSM signed messages

The GetDeviceStateRequest message from a TSM to a TEE shall include OCSP stapling data for the TSM's signer certificate and that for intermediate CA certificates up to the root certificate so that the TEE side can verify the signer certificate's revocation status.

A cCertificate revocation status check on a TA signer certificate is optional OPTIONAL by a TEE. A TSM is generally expected to do proper TA application vetting and its SP signer trust validation. A TEE will trust a TA signer certificate's validation status done by a TSM when it trusts the TSM.

Commented [DT132]: I can't parse this phrase.

15.10. Data protection at TSM and TEE

The TEE implementation provides protection of data on the device. It is the responsibility of the TSM to protect data on its servers.

15.11. Privacy consideration

Devices are issued with a unique TEE certificate to attest the~~a~~ device's validity. This uniqueness also creates a privacy and tracking risk that must be mitigated.

The TEE will only release the TEE certificate to a trusted TSM (it must verify the TSM certificate before proceeding). ~~The~~ OTrP ~~protocol~~ is designed such that only ~~the~~ a TSM can obtain the TEE device certificate and firmware certificate - the GetDeviceState message requires signature checks to validate the TSM is trusted, and ~~then~~OTrP delivers the device's certificate(s) encrypted such that only that TSM ~~may~~ can decrypt the response. A Client Application will never see the device certificate.

An SP~~–~~-specific TEE SP AIK (TEE SP Anonymous Key) is generated by the protocol for Client Applications. This provides a way for the Client Application to validate data sent from the TEE without requiring the TEE device certificate to be released to the client device rich O/S~~–~~, and to optionally allow an SP to encrypt a TA for a target device without the SP needing to be supplied with the TEE device certificate.

Commented [DT133]: One AIK for all of them, or one for each of them?

Commented [DT134]: Like what?

15.12. Threat mitigation

A rogue application may perform excessive TA loading. An OTrP Agent implementation should protect against excessive calls.

Commented [DT135]: Since the OTrP Agent is also untrusted, this doesn't make sense to me.

Rogue applications ~~may~~ might request excessive SD creation~~ request~~. The TSM is responsible to ensure this is properly guarded against.

Rogue OTrP Agents could replay or send TSM messages out of sequence: e.g., a TSM sends update1 and update2. The OTrP Agent replays update2 and update1 again, creating~~e~~ an unexpected result that a client wants. "dsihash" is used to mitigate this. The TEE MUST ~~make sure it~~ store~~s~~ the DSI state and check~~s~~ that the DSI state matches before it does another update.

Concurrent calls from a TSM to a TEE should be handled properly by a TEE. It is up to the device to manage concurrency to the TEE. If multiple concurrent TSM operations take place, these could fail due to the "dsihash" being modified by another concurrent operation. If locking is implemented on the client, this must be done in such a way that one application cannot lock other applications from using the TEE, except

Commented [DT136]: MUST?

Commented [DT137]: Meaning what?

Commented [DT138]: Meaning what? In an untrusted client app??
Seems like the TEE is the only entity that can do this mitigation.

for a short-~~-~~term duration of the TSM operation taking place. For example, an OTrP operation that starts but never completes (e.g., loss of connectivity) must not prevent subsequent OTrP messages from being executed.

15.13. Compromised CA

If a root CA for TSM certificates is ~~found~~ compromised, some TEE trust anchor update mechanism should be devised. A compromised intermediate CA is covered by OCSP stapling and the OCSP validation check in the protocol. A TEE should validate certificate revocation about a TSM certificate chain.

If the root CA of some TEE device certificates is compromised, these devices might be rejected by a TSM, which is a decision of the TSM implementation and a policy choice. Any intermediate CA for TEE device certificates should be validated by a TSM with a common CRL or OCSP method.

15.14. Compromised TSM

The TEE should use validation of the supplied TSM certificates and OCSP stapled data to validate that the TSM is trustworthy.

Since PKI is used, the integrity of the clock within the TEE determines the ability of the TEE to reject an expired TSM certificate, or revoked TSM certificate. Since OCSP stapling includes signature generation time, certificate validity dates are compared to the current time.

15.15. Certificate renewal

TFW and TEE device certificates are expected to be long-~~-~~lived, longer than the lifetime of a device. A TSM certificate usually has a moderate lifetime of 2 to 5 years. A TSM should get renewed or rekeyed certificates. The root CA certificates for a TSM, which ~~is~~ are embedded into the trust anchor store in a device, should have long lifetimes that don't require device trust anchor updates. On the other hand, it is imperative that OEMs or device providers plan for support of trust anchor update in their shipped devices.

16. References

16.1. Normative References

Commented [DT139]: This sounds like a TODO for the authors of this document.

Commented [DT140]: Can't parse this phrase.

Commented [DT141]: MUST?

Commented [DT142]: Which one specifically is mandated?

Commented [DT143]: Expand acronym on first use.

Commented [DT144]: MUST?

Commented [DT145]: So is secure absolute time mandatory? Is there a normative dependency on something like Secure NTP?

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

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <http://www.rfc-editor.org/info/rfc7516>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <http://www.rfc-editor.org/info/rfc7517>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <http://www.rfc-editor.org/info/rfc7518>.

16.2. Informative References

[GPTEE] Global Platform, "Global Platform, GlobalPlatform Device Technology: TEE System Architecture, v1.0", 2013.

Appendix A. Sample Messages

A.1. Sample Security Domain Management Messages

A.1.1. Sample GetDeviceState

A.1.1.1. Sample GetDeviceStateRequest

The TSM builds a "GetDeviceStateTBSRequest" message.


```
{
    "GetDeviceStateTBSRequest": {
      "ver": "1.0",
      "rid": "8C6F9DBB-FC39-435c-BC89-4D3614DA2F0B",
      "tid": "4F454A7F-002D-4157-884E-B0DD1A06A8AE",
      "ocspdat": "c2FtcGxlIG9jc3BkYXQgQjY0IGVuY29kZWQgQVNOMQ==",
      "icaocspdat": "c2FtcGxlIGljYW9jc3BkYXQgQjY0IGVuY29kZWQgQVNOMQ==",
      "supportedsigalgs": "RS256"
    }
}
```

The TSM signs "GetDeviceStateTBSRequest", creating "GetDeviceStateRequest".


```
{
  "GetDeviceStateRequest": {
    "payload":"
    ewoJIkdldERldmljZVN0YXRlVEJTUmVxdWVzdCI6IHsKCQkidmVyIjogIjEuMCIsCgkJ
    InJpZCI6IHs4QzZGOURCQi1GQzM5LTQzNWMtQkM4OS00RDM2MTREQTJGMEJ9LAoJCSJ0
    aWQiOiAiezRGNDU0QTdGLTAwMkQtNDE1Ny04ODRFLUIwREQxQTA2QThBRX0iLAoJCSJv
    Y3NwZGF0IjogImMyRnRjR3hsSUc5amMzQmtZWFFnUWpZMElHVnVZMjlrWldRZ1FWTk9N
    UT09IiwKCQkiaWNhb2NzcGRhdCI6ICJjMkZ0Y0d4bElHbGpZVzlqYzNCa1lYUWdRalkw
    SUdWdVkyOWtaV1FnUVZOT01RPT0iLAoJCSJzdXBwb3J0ZWRzaWdhbGdzIjogIlJTMjU2
    IgoJfQp9",
    "protected": "eyJhbGciOiJSUzI1NiJ9",
    "header": {
      "x5c": ["ZXhhbXBsZSBBU04xIHNpZ25lciBjZXJ0aWZpY2F0ZQ==",
              "ZXhhbXBsZSBBU04xIENBIGNlcnRpZmljYXRl"]
    },
    "signature":"c2FtcGxlIHNpZ25hdHVyZQ"
  }
}
```


A.1.1.2. Sample GetDeviceStateResponse

The TSM sends a "GetDeviceStateRequest" to the OTrP Agent.

The OTrP Agent obtains "dsi" from each TEE. (In this example there is a single TEE.)

The TEE obtains signed "fwdata" from firmware.

The TEE builds "dsi" - summarizing device state of the TEE.

```
{
  "dsi": {
    "tfwdata": {
      "tbs": "ezRGNDU0QTdGLTAwMkQtNDE1Ny04ODRFLUIwREQxQTA2QThBRX0=",
      "cert": "ZXhhbXBsZSBGVyBjZXJ0aWZpY2F0ZQ==",
      "sigalg": "RS256",
      "sig": "c2FtcGxlIEZXIHNpZ25hdHVyZQ=="
    },
    "tee": {
      "name": "Primary TEE",
      "ver": "1.0",
      "cert": "c2FtcGxlIFRFRSBjZXJ0aWZpY2F0ZQ==",
      "cacert": [
        "c2FtcGxlIENBIGNlcnRpZmljYXRlIDE=",
        "c2FtcGxlIENBIGNlcnRpZmljYXRlIDI="
      ],
    "sdlist": {
      "cnt": "1",
      "sd": [
      {
        "name": "default.acmebank.com",
        "spid": "acmebank.com",
        "talist": [
          {
            "taid": "acmebank.secure.banking",
            "taname": "Acme secure banking app"
          },
          {
            "taid": "acmebank.loyalty.rewards",
            "taname": "Acme loyalty rewards app"
          }
        ]
      }
      ]
    },
    "teeaiklist": [
      {
        "spaik": "c2FtcGxlIEFTTjEgZW5jb2RlZCBQS0NTMSBwdWJsaWNrZXk=",
        "spaiktype": "RSA",
        "spid": "acmebank.com"
      }
    ]
    }
  }
}
```


The TEE encrypts "dsi", and embeds it into a "GetDeviceTEEStateTBSResponse" Message.

```
{
  "GetDeviceTEEStateTBSResponse": {
    "ver": "1.0",
    "status": "pass",
    "rid": "{8C6F9DBB-FC39-435c-BC89-4D3614DA2F0B}",
    "tid": "{4F454A7F-002D-4157-884E-B0DD1A06A8AE}",
    "signerreq":"false",
    "edsi": {
      "protected": "eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0K",
      "recipients": [
        {
          "header": {
          "alg": "RSA1_5"
        },
        "encrypted_key":
        "
        QUVTMTI4IChDRUspIGtleSwgZW5jcnlwdGVkIHdpdGggVFNNIFJTQSBwdWJsaWMg
        a2V5LCB1c2luZyBSU0ExXzUgcGFkZGluZw"
        }
      ],
      "iv": "ySGmfZ69YlcEilNr5_SGbA",
      "ciphertext":
      "
      c2FtcGxlIGRzaSBkYXRhIGVuY3J5cHRlZCB3aXRoIEFFUzEyOCBrZXkgZnJvbSByZW
      NpcGllbnRzLmVuY3J5cHRlZF9rZXk",
      "tag": "c2FtcGxlIGF1dGhlbnRpY2F0aW9uIHRhZw"
    }
  }
}
```


TEE signs "GetDeviceTEEStateTBSResponse" and returns it to the OTrP Agent. The OTrP Agent encodes "GetDeviceTEEStateResponse" into an array to form "GetDeviceStateResponse".

```
{
  "GetDeviceStateResponse": [
    {
      "GetDeviceTEEStateResponse": {
        "payload":
        "
        ewogICJHZXREZXZpY2VURUVTdGF0ZVRCU1Jlc3BvbnNlIjogewogICAgInZlciI6
        ICIxLjAiLAogICAgInN0YXR1cyI6ICJwYXNzIiwKICAgICJyaWQiOiAiezhDNkY5
        REJCLUZDMzktNDM1Yy1CQzg5LTREMzYxNERBMkYwQn0iLAogICAgInRpZCI6ICJ7
        NEY0NTRBN0YtMDAyRC00MTU3LTg4NEUtQjBERDFBMDZBOEFFfSIsCgkic2lnbmVy
        cmVxIjoiZmFsc2UiLAogICAgImVkc2kiOiB7CiAgICAgICJwcm90ZWN0ZWQiOiAi
        ZXlKbGJtTWlPaUpCTVRJNFEwSkRMVWhUTWpVMkluMEsiLAogICAgICAicmVjaXBp
        ZW50cyI6IFsKICAgICAgICB7CiAgICAgICAgICAiaGVhZGVyIjogewogICAgICAg
        ICAgImFsZyI6ICJSU0ExXzUiCiAgICAgICAgfSwKICAgICAgICAiZW5jcnlwdGVk
        X2tleSI6CiAgICAgICAgIgogICAgICAgIFFFVVlRNVEk0SUNoRFJVc3BJR3RsZVN3
        Z1pXNWpbmx3ZEdWa0lIZHBkR2dnVkZOTklGSlRRU0J3ZFdKc2FXTWcKICAgICAg
        ICBhMlY1TENMWMybHVaeUJTVTBFeFh6VWdjR0ZrWkdsdVp3IgogICAgICAgIH0K
        ICAgICAgXSwKICAgICAgIml2IjogInlTR21mWjY5WWxjRWlsTnI1X1NHYkEiLAog
        ICAgICAiY2lwaGVydGV4dCI6CiAgICAgICIKICAgICAgYzJGdGNHeGxJR1J6YVNC
        a1lYUmhJR1Z1WTNKNWNIUmxaQ0IzYVhSb0lFRkZVekV5T0NClpYa2dabkp2YlNC
        eVpXCiAgICAgIE5wY0dsbGJuUnpMbVZ1WTNKNWNIUmxaRjlyWlhrIiwKICAgICAg
        InRhZyI6ICJjMkZ0Y0d4bElHRjFkR2hsYm5ScFkyRjBhVzl1SUhSaFp3IgogICAg
        fQogIH0KfQ",
        "protected": "eyJhbGciOiJSUzI1NiJ9",
        "signature": "c2FtcGxlIHNpZ25hdHVyZQ"
      }
    }
  ]
}
```


The TEE returns "GetDeviceStateResponse" back to the OTrP Agent, which returns message back to the TSM.

A.1.2. Sample CreateSD

A.1.2.1. Sample CreateSDRequest

```
{
  "CreateSDTBSRequest": {
    "ver":"1.0",
    "rid":"req-01",
    "tid":"tran-01",
    "tee":"SecuriTEE",
    "nextdsi":"false",
    "dsihash":"Iu-c0-fGrpMmzbbtiWI1U8u7wMJE7IK8wkJpsVuf2js",
    "content":{
      "spid":"bank.com",
      "sdname":"sd.bank.com",
      "spcert":"MIIDFjCCAn-
      gAwIBAgIJAIk0Tat0tquDMA0GCSqGSIb3DQEBBQUAMGwxCzAJBgNVBAYTAktSMQ4wD
      AYDVQQIDAVTZW91bDESMBAGA1UEBwwJR3Vyby1kb25nMRAwDgYDVQQKDAdTb2xhY2l
      hMRAwDgYDVQQLDAdTb2xhY2lhMRUwEwYDVQQDDAxTb2xhLWNpYS5jb20wHhcNMTUwN
      zAyMDg1MTU3WhcNMjAwNjMwMDg1MTU3WjBsMQswCQYDVQQGEwJLUjEOMAwGA1UECAw
      FU2VvdWwxEjAQBgNVBAcMCUd1cm8tZG9uZzEQMA4GA1UECgwHU29sYWNpYTEQMA4GA
      1UECwwHU29sYWNpYTEVMBMGA1UEAwwMU29sYS1jaWEuY29tMIGfMA0GCSqGSIb3DQE
      BAQUAA4GNADCBiQKBgQDYWLrFf2OFMEciwSYsyhaLY4kslaWcXA0hCWJRaFzt5mU-
      lpSJ4jeu92inBbsXcI8PfRbaItsgW1TD1Wg4gQH4MX_YtaBoOepE--
      3JoZZyPyCWS3AaLYWrDmqFXdbzaO1i8GxB7zz0gWw55bZ9jyzcl5gQzWSqMRpx_dca
      d2SP2wIDAQABo4G_MIG8MIGGBgNVHSMEfzB9oXCkbjBsMQswCQYDVQQGEwJLUjEOMA
      wGA1UECAwFU2VvdWwxEjAQBgNVBAcMCUd1cm8tZG9uZzEQMA4GA1UECgwHU29sYWNp
      YTEQMA4GA1UECwwHU29sYWNpYTEVMBMGA1UEAwwMU29sYS1jaWEuY29tggkAiTRNq3
      S2q4MwCQYDVR0TBAIwADAOBgNVHQ8BAf8EBAMCBsAwFgYDVR0lAQH_BAwwCgYIKwYB
      BQUHAwMwDQYJKoZIhvcNAQEFBQADgYEAEFMhRwEQ-
      LDa9O7P1N0mcLORpo6fW3QuJfuXbRQRQGoXddXMKazI4VjbGaXhey7Bzvk6TZYDa-
      GRiZby1J47UPaDQR3UiDzVvXwCOU6S5yUhNJsW_BeMViYj4lssX28iPpNwLUCVm1QV
      THILI6afLCRWXXclc1L5KGY290OwIdQ",
      "tsmid":"tsm_x.acme.com",
      "did":"zAHkb0-SQh9U_OT8mR5dB-tygcqpUJ9_x07pIiw8WoM"
    }
  }
}
```


~~Here~~ Below is a sample message after the content is encrypted and encoded


```
{
  "CreateSDRequest": {
  "payload":"
  eyJDcmVhdGVTRFRCU1JlcXVlc3QiOnsidmVyIjoiMS4wIiwicmlkIjoicmVxLTAxIiwidG
  lkIjoidHJhbi0wMSIsInRlZSI6IlNlY3VyaVRFRSIsIm5leHRkc2kiOiJmYWxzZSIsImRz
  aWhhc2giOiIyMmVmOWNkM2U3YzZhZTkzMjZjZGI2ZWQ4OTYyMzU1M2NiYmJjMGMyNDRlYz
  gyYmNjMjQyNjliMTViOWZkYTNiIiwiY29udGVudCI6eyJwcm90ZWN0ZWQiOiJlLUtBbkdW
  dVktS0FuVHJpZ0p4Qk1USTRRMEpETFVoVE1qVTI0b0NkZlEiLCJyZWNpcGllbnRzIjpbey
  JoZWFkZXIiOnsiYWxnIjoiUlNBMV81In0sImVuY3J5cHRlZF9rZXkiOiJTUzE2NTl4Q2FJ
  c1dUeUlsVTZPLUVsZzU4UUhvT1pCekxVRGptVG9vanBaWE54TVpBakRMcWtaSTdEUzhOVG
  FIWHcxczFvZjgydVhsM0d6NlVWMkRoZDJ3R2l6Y2VEdGtXc1RwZDg4QVYwaWpEYTNXa3lk
```

```
  dEpSVmlPOGdkSlEtV29NSUVJRUxzVGthblZCb25wQkF4ZHE0ckVMbl9TZlliaFg4Zm9ub2
  gxUVUifV0sIml2IjoiQXhZOERDdERhR2xzYkdsamIzUm9aUSIsImNpcGhlcnRleHQiOiI1
  bmVWZXdndm55UXprR3hZeWw5QlFrZTJVNjVaOHp4NDdlb3NzM3FETy0xY2FfNEpFY3NLcj
  ZhNjF5QzBUb0doYnJOQWJXbVRSemMwSXB5bTF0ZjdGemp4UlhBaTZBYnVSM2gzSUpRS1Bj
  UUVvRUlkZ2tWX0NaZTM2eTBkVDBpRFBMclg0QzFkb0dmMEdvaWViRC1yVUg1VUtEY3BsTW
  9lTjZvUnFyd0dnNUhxLTJXM3B4MUlzY0h4SktRZm11dkYxMTJ4ajBmZFNZX0N2WFE1NTJr
  TVRDUW1ZbzRPaGF2R0ZvaG9TZVVnaGZSVG1LYWp3OThkTzdhREdrUEpRUlBtYVVHWllEMW
  JXd01nMXFRV3RPd19EZlIyZDNzTzVUN0pQMDJDUFprVXBiQ3dZYVcybW9HN1c2Zlc2U3V5
  Q2lpd2pQWmZSQmIzSktTVTFTd1kxYXZvdW02OWctaDB6by12TGZvbHRrWFV2LVdPTXZTY0
  JzR25NRzZYZnMzbXlTWnJ1WTNRR09wVVRzdjFCQ0JqSTJpdjkwb2U2aXFCcVpxQVBxbzdi
  ajYwVlJGQzZPTlNLZExGQTIyU3pqRHo1dmtnTXNEaHkwSzlDeVhYN1Z6MkNLTXJvQjNiUE
  xFZF9abTZuVWlkTFN5cVJ5cXJxTmVnN1lmQng3aV93X0dzRW9rX1VYZXd6RGtneHp6RjZj
  XzZ6S0s3UFktVnVmYUo0Z2dHZmlpOHEwMm9RZ1VEZTB2Vm1FWDc0c2VQX2RxakVpZVVOYm
  xBZE9sS2dBWlFGdEs4dy1xVUMzSzVGTjRoUG9yeDc2b3lPVUpOQTVFZVV2Qy1jR2tMcTNQ
  UG1GRmQyaUtOTElCTEJzVWl6c1h3RERvZVA5SmktWGt5ZEQtREN1SHdpcno0OEdNNWVLSj
  Q5WVdqRUtFQko2T01NNUNmZHZ4cDNmVG1uUTdfTXcwZ3FZVDRiOUJJSnBfWjA3TTctNUpE
  emg0czhyU3dsQzFXU3V2RmhRWlJCcXJtX2RaUlRIb0VaZldXc1VCSWVNWWdxNG1zb0JqTj
  NXSzhnRWYwZGI5a3Z6UG9LYmpJRy10UUE2R2l1X3pHaFVfLXFBV1lLemVKMDZ6djRIWlBO
  dHktQXRyTGF0WGhtUTdOQlVrX0hvbjdOUWxhU1g1ZHVNVmN4bGs1ZHVrWFZNMDgxa09wYV
  kzbDliQVFfYVhTM0FNaFFTTVVsT3dnTDZJazFPYVpaTGFMUE3ejlITnlESmFEWTVhakZK
  TWFDV1lfOG94YlNoQUktNXA2MmNuT0xzV0dNWWNKTlBGVTZpcWlMR19oc3JfNlNKMURhbD
  VtQ0YycnBJLUItMlhuckxZR01ZS0NEZ2V2dGFnbi1DVUV6RURwR3ozQ2VLcWdQU0Vqd3BK
  N0M3NXduYTlCSmtTUkpOdDNla3hoWElrcnNEazRHVVpMSDdQYzFYZHdRTXhxdWpzNmxJSV
  EycjM1NWEtVkotWHdPcFpfY3RPdW96LTA4WHdYQ3RkTEliSFFVTG40RjlMRTRtanU0dUxS
  bjNSc043WWZ1S3dCVmVEZDJ6R3NBY0s5SVlDa3hOaDk3dDluYW1iMDZqSXVoWXF5QkhWRU
  9nTkhici1rMDY1bW9OVk5lVVUyMm5OdVNKS0ZxVnIxT0dKNGVfNXkzYkNwTmxTeEFPV1Bn
  RnJzU0Flc2JJOWw4eVJtVTAwenJYdGc4OWt5SjlCcXN2eXA1RE8wX2FtS1JyMXB1MVJVWF
  lFZzB2ampKS1FSdDVZbXRUNFJzaWpqdGRDWDg3UUxJaUdSY0hDdlJzUzZSdDJESmNYR1ht
  UGQyc0ZmNUZyNnJnMkFzX3BmUHN3cnF1WlAxbVFLc3RPMFVkTXpqMTlyb2N1NHVxVXlHUD
  lWWU54cHVnWVdNSjRYb1dRelJtWGNTUEJ4VEtnenFPS2s3UnRzWWVMNXl4LVM4NjV0cHVz
  dTA0bXpzYUJRZ21od1ZFVXBRdWNrcG1YWkNLNHlJUXktaHNFQUlJSmVxdFB3dVAySXF0X2
  I5dlk0bzExeXdzeXhzdmp2RnNKN0VVZU1MaGE2R2dSanBSbnU5RWIzRnlJZ0U5M0VVNEEw
  T0lUMWlOSGNRYWc0eWtOc3dPdkxQbjZIZ21zQ05ESlgwekc2RlFDMTZRdjBSQ25SVTdfV2
  VvblhSTUZwUzZRZ1JiSk45R1NMckN5bklJSWxUcDBxNHBaS05zM0tqQ2tMUzJrb3Bhd2Y0
  WF9BUllmTko3a0s5eW5BR0dCcktnUWJNRWVxUEFmMDBKMlYtVXpuU1JMZmQ4SGs3Y2JEdk
  5RQlhHQW9BR0ViaGRwVUc0RXFwMlVyQko3dEtyUUVSRlh4RTVsOFNHY2czQ1RmN2Zoazdx
  VEFBVjVsWEFnOUtOUDF1c1ZRZk1fUlBleHFNTG9WQVVKV2syQkF6WF9uSEhkVVhaSVBIOG
  hLeDctdEFRV0dTWUd0R2FmanZJZzI2c082TzloQWZVd3BpSV90MzF6SkZORDU0OTZURHBz
  QmNnd2dMLU1UcVhCRUJ2NEhvQld5SG1DVjVFMUwiLCJ0YWciOiJkbXlEeWZJVlNJUi1Ren
  ExOEgybFRIeEMxbl9HZEtrdnZNMDJUcHdsYzQwIn19fQ",
  "protected":"e-KAnGFsZ-KAnTrigJxSUzI1NuKAnX0",    //RSAwithSHA256
  "header": {
    "kid":"e9bc097a-ce51-4036-9562-d2ade882db0d",
      "signer":"
      MIIC3zCCAkigAwIBAgIJAJf2fFkE1BYOMA0GCSqGSIb3DQEBBQUAMFoxCzAJBgNVBA
      YTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRMwEQYDVQQHDApDYWxpZm9ybmlhMSEw
      HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQwHhcNMTUwNzAyMDkwMTE4Wh
      cNMjAwNjMwMDkwMTE4WjBaMQswCQYDVQQGEwJVUzETMBEGA1UECAwKQ2FsaWZvcm5p
```

```
      YTETMBEGA1UEBwwKQ2FsaWZvcm5pYTEhMB8GA1UECgwYSW50ZXJuZXQgV2lkZ2l0cy
      BQdHkgTHRkMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC8ZtxM1bYickpgSVG-
      meHInI3f_chlMBdL8l7daOEztSs_a6GLqmvSu-
      AoDpTsfEd4EazdMBp5fmgLRGdCYMcI6bgpO94h5CCnlj8xFKPq7qGixdwGUA6b_ZI3
      c4cZ8eu73VMNrrn_z3WTZlExlpT9XVj-
      ivhfJ4a6T20EtMM5qwIDAQABo4GsMIGpMHQGA1UdIwRtMGuhXqRcMFoxCzAJBgNVBA
      YTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRMwEQYDVQQHDApDYWxpZm9ybmlhMSEw
      HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGSCCQCX9nxZBNQWDjAJBgNVHR
      MEAjAAMA4GA1UdDwEB_wQEAwIGwDAWBgNVHSUBAf8EDDAKBggrBgEFBQcDAzANBgkq
      hkiG9w0BAQUFAAOBgQAGkz9QpoxghZUWT4ivem4cIckfxzTBBiPHCjrrjB2X8Ktn8G
      SZ1MdyIZV8fwdEmD90IvtMHgtzK-
      9wo6Aibj_rVIpxGb7trP82uzc2X8VwYnQbuqQyzofQvcwZHLYplvi95pZ5fVrJvnYA
      UBFyfrdT5GjqL1nqH3a_Y3QPscuCjg"
        },
     "signature":"nuQUsCTEBLeaRzuwd7q1iPIYEJ2eJfurO5sT5Y-
     N03zFRcv1jvrqMHtx_pw0Y9YWjmpoWfpfelhwGEko9SgeeBnznmkZbp7kjS6MmX4CKz
     9OApe3-VI7yL9Yp0WNdRh3425eYfuapCy3lcXFln5JBAUnU_OzUg3RWxcU_yGnFsw"
  }
}
```


A.1.2.2. Sample CreateSDResponse


```
{
  "CreateSDTBSResponse": {
    "ver":"1.0",
    "status":"pass",
    "rid":"req-01",
    "tid":"tran-01",
    "content":{
      "did":"zAHkb0-SQh9U_OT8mR5dB-tygcqpUJ9_x07pIiw8WoM",
      "sdname":"sd.bank.com",
      "teespaik":"AQABjY9KiwH3hkMmSAAN6CLXot525U85WNlWKAQz5TOdfe_CM8h-
      X6_EHX1gOXoyRXaBiKMqWb0YZLCABTw1ytdXy2kWa525imRho8Vqn6HDGsJDZPDru9
      GnZR8pZX5ge_dWXB_uljMvDttc5iAWEJ8ZgcpLGtBTGLZnQoQbjtn1lIE",
    }
  }
}
```


~~Here~~ Below is the response message after the content is encrypted and encoded.


```
{
  "CreateSDResponse": {
    "payload":"
    eyJDcmVhdGVTRFRCU1Jlc3BvbnNlIjp7InZlciI6IjEuMCIsInN0YXR1cyI6InBhc3Mi
    LCJyaWQiOiJyZXEtMDEiLCJ0aWQiOiJ0cmFuLTAxIiwiY29udGVudCI6eyJwcm90ZWN0
    ZWQiOiJlLUtBbkdWdVktS0FuVHJpZ0p4Qk1USTRRMEpETFVoVE1qVTI0b0NkZlEiLCJy
    ZWNpcGllbnRzIjpbeyJoZWFkZXIiOnsiYWxnIjoiUlNBMV81In0sImVuY3J5cHRlZF9r
```

```
    ZXkiOiJOX0I4R3pldUlfN2hwd0wwTFpHSTkxVWVBbmxJRkJfcndmZU1yZERrWnFGak1s
    VVhjdlI0XzhhOGhyeFI4SXR3aEtFZnVfRWVLRDBQb0dqQ2pCSHcxdG1ULUN6eWhsbW5v
    Slk3LXllWnZzRkRpc2VNTkd0eGE0OGZJYUs2VWx5NUZMYXBCZVc5T1I5bmktOU9GQV9j
    aFVuWWl3b2Q4ZTJFa0Vpd0JEZ1EzMk0ifV0sIml2IjoiQXhZOERDdERhR2xzYkdsamIz
    Um9aUSIsImNpcGhlcnRleHQiOiJsalh6Wk5JTmR1WjFaMXJHVElkTjBiVUp1RDRVV2xT
    QVptLWd6YnJINFVDYy1jMEFQenMtMWdWSFk4NTRUR3VMYkdyRmVHcDFqM2Fsb1lacWZp
    ZnE4aEt3Ty16RFlBN2tmVFhBZHp6czM4em9xeG4zbHoyM2w1RUlGUWhrOHBRWTRYTHRW
    M3ZBQWlNYnlrQ1Q3VS1CWDdWcjBacVNhYWZTQVZ4OFBLQ1RIU3hHN3hHVko0NkxxRzJS
    RE54WXQ4RC1SQ3lZUi1zRTM0MUFKZldEc2FLaGRRbzJXcjNVN1hTOWFqaXJtWjdqTlJ4
    cVRodHJBRWlIY1ctOEJMdVFHWEZ1YUhLMTZrenJKUGl4d0VXbzJ4cmw4cmkwc3ZRcHpl
    Z2M3MEt2Z0I0NUVaNHZiNXR0YlUya25hN185QU1Wcm4wLUJaQ1Bnb280MWlFblhuNVJn
    TXY2c2V2Y1JPQ2xHMnpWSjFoRkVLYjk2akEiLCJ0YWciOiIzOTZISTk4Uk1NQnR0eDlo
    ZUtsODROaVZLd0lJSzI0UEt2Z1RGYzFrbEJzIn19fQ",
    "protected": "e-KAnGFsZ-KAnTrigJxSUzI1NuKAnX0",
    "header": {
        "kid":"e9bc097a-ce51-4036-9562-d2ade882db0d",
        "signer":"
            MIIC3zCCAkigAwIBAgIJAJf2fFkE1BYOMA0GCSqGSIb3DQEBBQUAMFoxCzAJ
            BgNVBAYTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRMwEQYDVQQHDApDYWxp
            Zm9ybmlhMSEwHwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQwHhcN
            MTUwNzAyMDkwMTE4WhcNMjAwNjMwMDkwMTE4WjBaMQswCQYDVQQGEwJVUzET
            MBEGA1UECAwKQ2FsaWZvcm5pYTETMBEGA1UEBwwKQ2FsaWZvcm5pYTEhMB8G
            A1UECgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIGfMA0GCSqGSIb3DQEB
            AQUAA4GNADCBiQKBgQC8ZtxM1bYickpgSVG-
            meHInI3f_chlMBdL8l7daOEztSs_a6GLqmvSu-
            AoDpTsfEd4EazdMBp5fmgLRGdCYMcI6bgpO94h5CCnlj8xFKPq7qGixdwGUA
            6b_ZI3c4cZ8eu73VMNrrn_z3WTZlExlpT9XVj-
            ivhfJ4a6T20EtMM5qwIDAQABo4GsMIGpMHQGA1UdIwRtMGuhXqRcMFoxCzAJ
            BgNVBAYTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRMwEQYDVQQHDApDYWxp
            Zm9ybmlhMSEwHwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGSCCQCX
            9nxZBNQWDjAJBgNVHRMEAjAAMA4GA1UdDwEB_wQEAwIGwDAWBgNVHSUBAf8E
            DDAKBggrBgEFBQcDAzANBgkqhkiG9w0BAQUFAAOBgQAGkz9QpoxghZUWT4iv
            em4cIckfxzTBBiPHCjrrjB2X8Ktn8GSZ1MdyIZV8fwdEmD90IvtMHgtzK-
            9wo6Aibj_rVIpxGb7trP82uzc2X8VwYnQbuqQyzofQvcwZHLYplvi95pZ5fV
            rJvnYAUBFyfrdT5GjqL1nqH3a_Y3QPscuCjg"
    },
    "signature":"jnJtaB0vFFwrE-qKOR3Pu9pf2gNoI1s67GgPCTq0U-
    qrz97svKpuh32WgCP2MWCoQPEswsEX-nxhIx_siTe4zIPO1nBYn-
    R7b25rQaF87O8uAOOnBN5Yl2Jk3laIbs-
    hGE32aRZDhrVoyEdSvIFrT6AQqD20bIAZGqTR-zA-900"
  }
}
```


A.1.3. Sample UpdateSD

A.1.3.1. Sample UpdateSDRequest


```
{
  "UpdateSDTBSRequest": {
    "ver": "1.0",
    "rid": "1222DA7D-8993-41A4-AC02-8A2807B31A3A",
    "tid": "4F454A7F-002D-4157-884E-B0DD1A06A8AE",
    "tee": "Primary TEE ABC",
    "nextdsi": "false",
    "dsihash":
    "
    IsOvwpzDk8Onw4bCrsKTJsONwrbDrcKJYjVTw4vCu8OAw4JEw6zCgsK8w4JCacKxW8Kf
    w5o7",
    "content": { // NEEDS to BE ENCRYPTED
      "tsmid": "id1.tsmxyz.com",
      "spid": "com.acmebank.spid1",
      "sdname": "com.acmebank.sdname1",
      "changes": {
        "newsdname": "com.acmebank.sdname2",
        "newspid": "com.acquirer.spid1",
        "spcert":
        "MIIDFjCCAn-
        gAwIBAgIJAIk0Tat0tquDMA0GCSqGSIb3DQEBBQUAMGwxCzAJBgNVBAYTAktSMQ4
        wDAYDVQQIDAVTZW91bDESMBAGA1UEBwwJR3Vyby1kb25nMRAwDgYDVQQKDAdTb2x
        hY2lhMRAwDgYDVQQLDAdTb2xhY2lhMRUwEwYDVQQDDAxTb2xhLWNpYS5jb20wHhc
        NMTUwNzAyMDg1MTU3WhcNMjAwNjMwMDg1MTU3WjBsMQswCQYDVQQGEwJLUjEOMAw
        GA1UECAwFU2VvdWwxEjAQBgNVBAcMCUd1cm8tZG9uZzEQMA4GA1UECgwHU29sYWN
        pYTEQMA4GA1UECwwHU29sYWNpYTEVMBMGA1UEAwwMU29sYS1jaWEuY29tMIGfMA0
        GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDYWLrFf2OFMEciwSYsyhaLY4kslaWcXA0
        hCWJRaFzt5mU-
        lpSJ4jeu92inBbsXcI8PfRbaItsgW1TD1Wg4gQH4MX_YtaBoOepE--
        3JoZZyPyCWS3AaLYWrDmqFXdbzaO1i8GxB7zz0gWw55bZ9jyzcl5gQzWSqMRpx_d
        cad2SP2wIDAQABo4G_MIG8MIGGBgNVHSMEfzB9oXCkbjBsMQswCQYDVQQGEwJLUj
        EOMAwGA1UECAwFU2VvdWwxEjAQBgNVBAcMCUd1cm8tZG9uZzEQMA4GA1UECgwHU2
        9sYWNpYTEQMA4GA1UECwwHU29sYWNpYTEVMBMGA1UEAwwMU29sYS1jaWEuY29tgg
        kAiTRNq3S2q4MwCQYDVR0TBAIwADAOBgNVHQ8BAf8EBAMCBsAwFgYDVR0lAQH_BA
        wwCgYIKwYBBQUHAwMwDQYJKoZIhvcNAQEFBQADgYEAEFMhRwEQ-
        LDa9O7P1N0mcLORpo6fW3QuJfuXbRQRQGoXddXMKazI4VjbGaXhey7Bzvk6TZYDa
        -
        GRiZby1J47UPaDQR3UiDzVvXwCOU6S5yUhNJsW_BeMViYj4lssX28iPpNwLUCVm1
        QVTHILI6afLCRWXXclc1L5KGY290OwIdQ",
        "renewteespaik": "0"
      }
    }
  }
}
```

A.1.3.2. Sample UpdateSDResponse


```
{
  "UpdateSDTBSResponse": {
    "ver": "1.0",
    "status": "pass",
    "rid": "1222DA7D-8993-41A4-AC02-8A2807B31A3A",
    "tid": "4F454A7F-002D-4157-884E-B0DD1A06A8AE",
    "content": {
      "did": "MTZENTE5Qzc0Qzk0NkUxMzYxNzk0NjY4NTc3OTY4NTI=",
      "teespaik":
      "AQABjY9KiwH3hkMmSAAN6CLXot525U85WNlWKAQz5TOdfe_CM8h-
      X6_EHX1gOXoyRXaBiKMqWb0YZLCABTw1ytdXy2kWa525imRho8Vqn6HDGsJDZPDru9
      GnZR8pZX5ge_dWXB_uljMvDttc5iAWEJ8ZgcpLGtBTGLZnQoQbjtn1lIE",
      "teespaiktype": "RSA"
    }
  }
}
```


A.1.4. Sample DeleteSD

A.1.4.1. Sample DeleteSDRequest

TSM builds a message - including data to be encrypted.


```
{
     "DeleteSDTBSRequest": {
       "ver": "1.0",
       "rid": "{712551F5-DFB3-43f0-9A63-663440B91D49}",
       "tid": "{4F454A7F-002D-4157-884E-B0DD1A06A8AE}",
       "tee": "Primary TEE",
       "nextdsi": "false",
       "dsihash": "AAECAwQFBgcICQoLDA0ODwABAgMEBQYHCAkKCwwNDg8=",
       "content": ENCRYPTED {
         "tsmid": "tsm1.com",
         "sdname": "default.acmebank.com",
         "deleteta": "1"
       }
     }
   }
```


TSM encrypts the "content".

```
{
  "DeleteSDTBSRequest": {
    "ver": "1.0",
    "rid": "{712551F5-DFB3-43f0-9A63-663440B91D49}",
    "tid": "{4F454A7F-002D-4157-884E-B0DD1A06A8AE}",
    "tee": "Primary TEE",
    "nextdsi": "false",
    "dsihash": "AAECAwQFBgcICQoLDA0ODwABAgMEBQYHCAkKCwwNDg8=",
    "content": {
    "protected": "eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
    "recipients": [
      {
        "header": {
          "alg": "RSA1_5"
        },
      "encrypted_key":
      "
      QUVTMTI4IChDRUspIGtleSwgZW5jcnlwdGVkIHdpdGggVFNNIFJTQSBwdWJsaWMga2
      V5LCB1c2luZyBSU0ExXzUgcGFkZGluZw"
      }
    ],
    "iv": "rWO5DVmQX9ogelMLBIogIA",
    "ciphertext":
    "
    c2FtcGxlIGRzaSBkYXRhIGVuY3J5cHRlZCB3aXRoIEFFUzEyOCBrZXkgZnJvbSByZWNp
    cGllbnRzLmVuY3J5cHRlZF9rZXk",
    "tag": "c2FtcGxlIGF1dGhlbnRpY2F0aW9uIHRhZw"
    }
  }
}
```


TSM signs "DeleteSDTBSRequest" to form "DeleteSDRequest"

```
{
  "DeleteSDRequest": {
    "payload":"
    ewoJIkRlbGV0ZVNEVEJTUmVxdWVzdCI6IHsKCQkidmVyIjogIjEuMCIsCgkJInJp
    ZCI6ICJ7NzEyNTUxRjUtREZCMy00M2YwLTlBNjMtNjYzNDQwQjkxRDQ5fSIsCgkJ
    InRpZCI6ICJ7NEY0NTRBN0YtMDAyRC00MTU3LTg4NEUtQjBERDFBMDZBOEFFfSIs
    CgkJInRlZSI6ICJQcmltYXJ5IFRFRSIsCgkJIm5leHRkc2kiOiAiZmFsc2UiLAoJ
    CSJkc2loYXNoIjogIkFBRUNBd1FGQmdjSUNRb0xEQTBPRHdBQkFnTUVCUVlIQ0Fr
    S0N3d05Ezg9IiwKCQkiY29udGVudCI6IHsKCQkJInByb3RlY3RlZCI6ICJleUps
    Ym1NaU9pSkJNVEk0UTBKRExVaFRNalUySW4wIiwKCQkJInJlY2lwaWVudHMiOiBb
    ewoJCQkJImhlYWRlciI6IHsKCQkJCQkiYWxnIjogIlJTQTFfNSIKCQkJCX0sCgkJ
    CQkiZW5jcnlwdGVkX2tleSI6ICJRVVZUTVRJNElDaERSVXNwSUd0bGVTd2daVzVq
    Y25sd2RHVmtJSGRwZEdnZ1ZGTk5JRkpUUVNCd2RXSnNhV01nYTJWNUxDQjFjMmx1
    WnlCU1UwRXhYelVnY0dGa1pHbHVadyIKCQkJfV0sCgkJCSJpdiI6ICJyV081RFZt
    UVg5b2dlbE1MQklvZ0lBIiwKCQkJImNpcGhlcnRleHQiOiAiYzJGdGNHeGxJR1J6
    YVNCa1lYUmhJR1Z1WTNKNWNIUmxaQ0IzYVhSb0lFRkZVekV5T0NClpYa2dabkp2
    YlNCeVpXTnBjR2xsYm5SekxtVnVZM0o1Y0hSbFpGOXJaWGsiLAoJCQkidGFnIjog
    ImMyRnRjR3hsSUdGMWRHaGxibllwWTJGMGFXOXVJSFJoWnciCgkJfQoJfQp9",
    "protected":"eyJhbGciOiJSUzI1NiJ9",
    "header": {
      "x5c": ["ZXhhbXBsZSBBU04xIHNpZ25lciBjZXJ0aWZpY2F0ZQ==",
              "ZXhhbXBsZSBBU04xIENBIGNlcnRpZmljYXRl"]
    },
    "signature":"c2FtcGxlIHNpZ25hdHVyZQ"
  }
}
```


A.1.4.2. Sample DeleteSDResponse

TEE creates "DeleteSDTBSResponse" to respond to the "DeleteSDRequest" message from the TSM, including data to be encrypted.


```
{
   "DeleteSDTBSResponse": {
     "ver": "1.0",
     "status": "pass",
     "rid": "{712551F5-DFB3-43f0-9A63-663440B91D49}",
     "tid": "{4F454A7F-002D-4157-884E-B0DD1A06A8AE}",
     "content": ENCRYPTED {
       "did": "MTZENTE5Qzc0Qzk0NkUxMzYxNzk0NjY4NTc3OTY4NTI=",
     }
   }
 }
```


TEE encrypts the "content" for the TSM.

```
{
 "DeleteSDTBSResponse": {
  "ver": "1.0",
  "status": "pass",
  "rid": "{712551F5-DFB3-43f0-9A63-663440B91D49}",
  "tid": "{4F454A7F-002D-4157-884E-B0DD1A06A8AE}",
   "content": {
   "protected": "eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0K",
   "recipients": [
     {
       "header": {
       "alg": "RSA1_5"
     },
     "encrypted_key":
     "
     QUVTMTI4IChDRUspIGtleSwgZW5jcnlwdGVkIHdpdGggVFNNIFJTQSBwdWJsaWMg
     a2V5LCB1c2luZyBSU0ExXzUgcGFkZGluZw"
     }
   ],
   "iv": "ySGmfZ69YlcEilNr5_SGbA",
   "ciphertext":
   "
   c2FtcGxlIGRzaSBkYXRhIGVuY3J5cHRlZCB3aXRoIEFFUzEyOCBrZXkgZnJvbSByZW
   NpcGllbnRzLmVuY3J5cHRlZF9rZXk",
   "tag": "c2FtcGxlIGF1dGhlbnRpY2F0aW9uIHRhZw"
   }
  }
}
```


TEE signs "DeleteSDTBSResponse" to form "DeleteSDResponse"

```
{
  "DeleteSDResponse": {
    "payload":"
    ewoJIkRlbGV0ZVNEVEJTUmVzcG9uc2UiOiB7CgkJInZlciI6ICIxLjAiLAoJCSJz
    dGF0dXMiOiAicGFzcyIsCgkJInJpZCI6ICJ7NzEyNTUxRjUtREZCMy00M2YwLTlB
    NjMtNjYzNDQwQjkxRDQ5fSIsCgkJInRpZCI6ICJ7NEY0NTRBN0YtMDAyRC00MTU3
    LTg4NEUtQjBERDFBMDZBOEFFfSIsCgkJImNvbnRlbnQiOiB7CgkJCSJwcm90ZWN0
    ZWQiOiAiZXlKbGJtTWlPaUpCTVRJNFEwSkRMVWhUTWpVMkluMEsiLAoJCQkicmVj
    aXBpZW50cyI6IFt7CgkJCQkiaGVhZGVyIjogewoJCQkJCSJhbGciOiAiUlNBMV81
    IgoJCQkJfSwKCQkJCSJlbmNyeXB0ZWRfa2V5IjogIlFVVlRNVEk0SUNoRFJVc3BJ
    R3RsZVN3Z1pXNWpjbmx3ZEdWa0lIZHBkR2dnVkZOTklGSlRRU0J3ZFdKc2FXTWdh
    MlY1TENCMWMybHVaeUJTVTBFeFh6VWdjR0ZrWkdsdVp3IgoJCQl9XSwKCQkJIml2
    IjogInlTR21mWjY5WWxjRWlsTnI1X1NHYkEiLAoJCQkiY2lwaGVydGV4dCI6ICJj
    MkZ0Y0d4bElHUnphU0JrWVhSaElHVnVZM0o1Y0hSbFpDQjNhWFJvSUVGRlV6RXlP
    Q0JyWlhrZ1puSnZiU0J5WldOcGNHbGxiblJ6TG1WdVkzSjVjSFJsWkY5clpYayIs
    CgkJCSJ0YWciOiAiYzJGdGNHeGxJR0YxZEdobGJuUnBZMkYwYVc5dUlIUmhadyIK
    CQl9Cgl9Cn0",
    "protected":"eyJhbGciOiJSUzI1NiJ9",
    "signature":"c2FtcGxlIHNpZ25hdHVyZQ"
  }
}
```


TEE returns "DeleteSDResponse" back to the OTrP Agent, which returns the message back to the TSM.

A.2. Sample TA Management Messages

A.2.1. Sample InstallTA

A.2.1.1. Sample InstallTARequest

```
{
  "InstallTATBSRequest": {
    "ver": "1.0",
    "rid": "24BEB059-0AED-42A6-A381-817DFB7A1207",
    "tid": "4F454A7F-002D-4157-884E-B0DD1A06A8AE",
    "tee": "Primary TEE ABC",
    "nextdsi": "true",
    "dsihash":
    "
    IsOvwpzDk8Onw4bCrsKTJsONwrbDrcKJYjVTw4vCu8OAw4JEw6zCgsK8w4JCacKxW8Kf
    w5o7",
    "content": {
      "tsmid": "id1.tsmxyz.com",
      "spid": "com.acmebank.spid1",
      "sdname": "com.acmebank.sdname1",
      "taid": "com.acmebank.taid.banking"
    },
    "encrypted_ta": {
      "key":
      "mLBjodcE4j36y64nC/nEs694P3XrLAOokjisXIGfs0H7lOEmT5FtaNDYEMcg9RnE
      ftlJGHO7N0lgcNcjoXBmeuY9VI8xzrsZM9gzH6VBKtVONSx0aw5IAFkNcyPZwDdZ
      MLwhvrzPJ9Fg+bZtrCoJz18PUz+5aNl/dj8+NM85LCXXcBlZF74btJer1Mw6ffzT
      /grPiEQTeJ1nEm9F3tyRsvcTInsnPJ3dEXv7sJXMrhRKAeZsqKzGX4eiZ3rEY+FQ
      6nXULC8cAj5XTKpQ/EkZ/iGgS0zcXR7KUJv3wFEmtBtPD/+ze08NILLmxM8olQFj
      //Lq0gGtq8vPC8r0oOfmbQ==",
      "iv": "4F5472504973426F726E496E32303135",
      "alg": "AESCBC",
      "ciphertadata":
      "......0x/5KGCXWfg1Vrjm7zPVZqtYZ2EovBow+7EmfOJ1tbk......=",
      "cipherpdata": "0x/5KGCXWfg1Vrjm7zPVZqtYZ2EovBow+7EmfOJ1tbk="
    }
  }
}
```


A.2.1.2. Sample InstallTAResponse

A sample to-be-signed response of InstallTA looks as follows.


```
{
  "InstallTATBSResponse": {
    "ver": "1.0",
    "status": "pass",
    "rid": "24BEB059-0AED-42A6-A381-817DFB7A1207",
    "tid": "4F454A7F-002D-4157-884E-B0DD1A06A8AE",
    "content": {
      "did": "MTZENTE5Qzc0Qzk0NkUxMzYxNzk0NjY4NTc3OTY4NTI=",
      "dsi": {
        "tfwdata": {
```

```
          "tbs": "ezRGNDU0QTdGLTAwMkQtNDE1Ny04ODRFLUIwREQxQTA2QThBRX0="
          "cert": "ZXhhbXBsZSBGVyBjZXJ0aWZpY2F0ZQ==",
          "sigalg": "UlMyNTY=",
          "sig": "c2FtcGxlIEZXIHNpZ25hdHVyZQ=="
        },
        "tee": {
          "name": "Primary TEE",
          "ver": "1.0",
          "cert": "c2FtcGxlIFRFRSBjZXJ0aWZpY2F0ZQ==",
          "cacert": [
            "c2FtcGxlIENBIGNlcnRpZmljYXRlIDE=",
            "c2FtcGxlIENBIGNlcnRpZmljYXRlIDI="
          ],
          "sdlist": {
            "cnt": "1",
            "sd": [
              {
                "name": "com.acmebank.sdname1",
                "spid": "com.acmebank.spid1",
                "talist": [
                    {
                    "taid": "com.acmebank.taid.banking",
                    "taname": "Acme secure banking app"
                    },
                    {
                    "taid": "acom.acmebank.taid.loyalty.rewards",
                    "taname": "Acme loyalty rewards app"
                    }
                ]
              }
            ]
          },
          "teeaiklist": [
            {
              "spaik":
                "c2FtcGxlIEFTTjEgZW5jb2RlZCBQS0NTMSBwdWJsaWNrZXk=",
              "spaiktype": "RSA"
              "spid": "acmebank.com"
            }
          ]
        }
      }
    }
  }
 }
```

A.2.2. Sample UpdateTA

A.2.2.1. Sample UpdateTARequest


```
{
  "UpdateTATBSRequest": {
    "ver": "1.0",
    "rid": "req-2",
    "tid": "tran-01",
    "tee": "SecuriTEE",
                "nextdsi": " false",
    "dsihash": "gwjul_9MZks3pqUSN1-eL1aViwGXNAxk0AIKW79dn4U",
    "content": {
      "tsmid": "tsm1.acme.com",
      "spid": "bank.com",
      "sdname": "sd.bank.com",
      "taid": "sd.bank.com.ta"
    },
    "encrypted_ta": {
      "key":
      "
      XzmAn_RDVk3IozMwNWhiB6fmZlIs1YUvMKlQAv_UDoZ1fvGGsRGo9bT0A440aYMgLt
      GilKypoJjCgijdaHgamaJgRSc4Je2otpnEEagsahvDNoarMCC5nGQdkRxW7Vo2NKgL
      A892HGeHkJVshYm1cUlFQ-BhiJ4NAykFwlqC_oc",
      "iv": "AxY8DCtDaGlsbGljb3RoZQ",
      "alg": "AESCBC",
      "ciphernewtadata":
      "KHqOxGn7ib1F_14PG4_UX9DBjOcWkiAZhVE-U-
      67NsKryHGokeWr2spRWfdU2KWaaNncHoYGwEtbCH7XyNbOFh28nzwUmstep4nHWbAl
      XZYTNkENcABPpuw_G3I3HADo"
    }
  }
}

{
  "UpdateTARequest": {
    "payload" :
    "
    eyJVcGRhdGVUQVRCU1JlcXVlc3QiOnsidmVyIjoiMS4wIiwicmlkIjoicmVxLTIiLCJ0
    aWQiOiJ0cmFuLTAxIiwidGVlIjoiU2VjdXJpVEVFIiwibmV4dGRzaSI6ImZhbHNlIiwi
    ZHNpaGFzaCI6Imd3anVsXzlNWmtzM3BxVVNOMS1lTDFhVml3R1hOQXhrMEFJS1c3OWRu
    NFUiLCJjb250ZW50Ijp7InByb3RlY3RlZCI6ImV5SmxibU1pT2lKQk1USTRRMEpETFVo
    VE1qVTJJbjAiLCJyZWNpcGllbnRzIjpbeyJoZWFkZXIiOnsiYWxnIjoiUlNBMV81In0s
    ImVuY3J5cHRlZF9rZXkiOiJYem1Bbl9SRFZrM0lvek13TldoaUI2Zm1abElzMVlVdk1L
    bFFBdl9VRG9aMWZ2R0dzUkdvOWJUMEE0NDBhWU1nTHRHaWxLeXBvSmpDZ2lqZGFIZ2Ft
    YUpnUlNjNEplMm90cG5FRWFnc2FodkROb2FyTUNDNW5HUWRrUnhXN1ZvMk5LZ0xBODky
    SEdlSGtKVnNoWW0xY1VsRlEtQmhpSjROQXlrRndscUNfb2MifV0sIml2IjoiQXhZOERD
    dERhR2xzYkdsamIzUm9aUSIsImNpcGhlcnRleHQiOiJIYTcwVXRZVEtWQmtXRFJuMi0w
```

```
    SF9IdkZtazl5SGtoVV91bk1OLWc1T3BqLWF1NGFUb2lxWklMYzVzYTdENnZZSjF6eW04
    QW1JOEJIVXFqc2l5Z0tOcC1HdURJUjFzRXc0a2NhMVQ5ZENuU0RydHhSUFhESVdrZmt3
    azZlR1NQWiIsInRhZyI6Im9UN01UTE41eWtBTFBoTDR0aUh6T1pPTGVFeU9xZ0NWaEM5
    MXpkcldMU0UifSwiZW5jcnlwdGVkX3RhIjp7ImtleSI6Ilh6bUFuX1JEVmszSW96TXdO
    V2hpQjZmbVpsSXMxWVV2TUtsUUF2X1VEb1oxZnZHR3NSR285YlQwQTQ0MGFZTWdMdEdp
    bEt5cG9KakNnaWpkYUhnYW1hSmdSU2M0SmUyb3RwbkVFYWdzYWh2RE5vYXJNQ0M1bkdR
    ZGtSeFc3Vm8yTktnTEE4OTJIR2VIa0pWc2hZbTFjVWxGUS1CaGlKNE5BeWtGd2xxQ19v
    YyIsIml2IjoiQXhZOERDdERhR2xzYkdsamIzUm9aUSIsImFsZyI6IkFFU0NCQyIsImNp
    cGhlcm5ld3RhZGF0YSI6IktIcU94R243aWIxRl8xNFBHNF9VWDlEQmpPY1draUFaaFZF
    LVUtNjdOc0tyeUhHb2tlV3Iyc3BSV2ZkVTJLV2FhTm5jSG9ZR3dFdGJDSDdYeU5iT0Zo
    MjhuendVbXN0ZXA0bkhXYkFsWFpZVE5rRU5jQUJQcHV3X0czSTNIQURvIn19fQ",
    "protected": " eyJhbGciOiJSUzI1NiJ9",
    "header": {
      "kid":"e9bc097a-ce51-4036-9562-d2ade882db0d",
      "signer":"
      MIIC3zCCAkigAwIBAgIJAJf2fFkE1BYOMA0GCSqGSIb3DQEBBQUAMFoxCzAJBgNVBA
      YTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRMwEQYDVQQHDApDYWxpZm9ybmlhMSEw
      HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQwHhcNMTUwNzAyMDkwMTE4Wh
      cNMjAwNjMwMDkwMTE4WjBaMQswCQYDVQQGEwJVUzETMBEGA1UECAwKQ2FsaWZvcm5p
      YTETMBEGA1UEBwwKQ2FsaWZvcm5pYTEhMB8GA1UECgwYSW50ZXJuZXQgV2lkZ2l0cy
      BQdHkgTHRkMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC8ZtxM1bYickpgSVG-
      meHInI3f_chlMBdL8l7daOEztSs_a6GLqmvSu-
      AoDpTsfEd4EazdMBp5fmgLRGdCYMcI6bgpO94h5CCnlj8xFKPq7qGixdwGUA6b_ZI3
      c4cZ8eu73VMNrrn_z3WTZlExlpT9XVj-
      ivhfJ4a6T20EtMM5qwIDAQABo4GsMIGpMHQGA1UdIwRtMGuhXqRcMFoxCzAJBgNVBA
      YTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRMwEQYDVQQHDApDYWxpZm9ybmlhMSEw
      HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGSCCQCX9nxZBNQWDjAJBgNVHR
      MEAjAAMA4GA1UdDwEB_wQEAwIGwDAWBgNVHSUBAf8EDDAKBggrBgEFBQcDAzANBgkq
      hkiG9w0BAQUFAAOBgQAGkz9QpoxghZUWT4ivem4cIckfxzTBBiPHCjrrjB2X8Ktn8G
      SZ1MdyIZV8fwdEmD90IvtMHgtzK-
      9wo6Aibj_rVIpxGb7trP82uzc2X8VwYnQbuqQyzofQvcwZHLYplvi95pZ5fVrJvnYA
      UBFyfrdT5GjqL1nqH3a_Y3QPscuCjg"
    },
    "signature":"inB1K6G3EAhF-
    FbID83UI25R5Ao8MI4qfrbrmf0UQhjM3O7_g3l6XxN_JkHrGQaZr-
    myOkGPVM8BzbUZW5GqxNZwFXwMeaoCjDKc4Apv4WZkD1qKJxkg1k5jaUCfJz1Jmw_XtX
    6MHhrLh9ov03S9PtuT1VAQ0FVUB3qFIvjSnNU"
  }
}
```


A.2.2.2. Sample UpdateTAResponse

```
{
  "UpdateTATBSResponse": {
    "ver": "1.0",
    "status": "pass",
        "rid": "req-2",
        "tid": "tran-01",
        "content": {
      "did": "zAHkb0-SQh9U_OT8mR5dB-tygcqpUJ9_x07pIiw8WoM"
    }
  }
}
```

```
{
  "UpdateTAResponse":{
    "payload":"
    eyJVcGRhdGVUQVRCU1Jlc3BvbnNlIjp7InZlciI6IjEuMCIsInN0YXR1cyI6InBhc3Mi
    LCJyaWQiOiJyZXEtMiIsInRpZCI6InRyYW4tMDEiLCJjb250ZW50Ijp7InByb3RlY3Rl
    ZCI6ImV5SmxibU1pT2lKQk1USTRRMEpETFVoVE1qVTJJbjAiLCJyZWNpcGllbnRzIjpb
    eyJoZWFkZXIiOnsiYWxnIjoiUlNBMV81In0sImVuY3J5cHRlZF9rZXkiOiJFaGUxLUJB
    UUdJLTNEMFNHdXFGY01MZDJtd0gxQm1uRndYQWx1M1FxUFVXZ1RRVm55SUowNFc2MnBK
    YWVSREFkeTU0R0FSVjBrVzQ0RGw0MkdUUlhqbE1EZ3BYdXdFLWloc1JVV0tNNldCZ2N3
    VXVGQTRUR3gwU0I1NTZCdl92dnBNaFdfMXh2c2FHdFBaQmwxTnZjbXNibzBhY3FobXlu
    bzBDTmF5SVAtX1UifV0sIml2IjoiQXhZOERDdERhR2xzYkdsamIzUm9aUSIsImNpcGhl
    cnRleHQiOiJwc2o2dGtyaGJXM0lmVElMeE9GMU5HdFUtcTFmeVBidV9KWk9jbklycWIw
    eTNPOHN6OTItaWpWR1ZyRW5WbG1sY1FYeWFNZTNyX1JGdEkwV3B4UmRodyIsInRhZyI6
    Ik0zb2dNNk11MVJYMUMybEZvaG5rTkN5b25qNjd2TDNqd2RrZXhFdUlpaTgifX19",
    "protected":"eyJhbGciOiJSUzI1NiJ9",
    "header": {
      "kid":"e9bc097a-ce51-4036-9562-d2ade882db0d",
      "signer":"
      MIIC3zCCAkigAwIBAgIJAJf2fFkE1BYOMA0GCSqGSIb3DQEBBQUAMFoxCzAJBgNVBA
      YTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRMwEQYDVQQHDApDYWxpZm9ybmlhMSEw
      HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQwHhcNMTUwNzAyMDkwMTE4Wh
      cNMjAwNjMwMDkwMTE4WjBaMQswCQYDVQQGEwJVUzETMBEGA1UECAwKQ2FsaWZvcm5p
      YTETMBEGA1UEBwwKQ2FsaWZvcm5pYTEhMB8GA1UECgwYSW50ZXJuZXQgV2lkZ2l0cy
      BQdHkgTHRkMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC8ZtxM1bYickpgSVG-
      meHInI3f_chlMBdL8l7daOEztSs_a6GLqmvSu-
      AoDpTsfEd4EazdMBp5fmgLRGdCYMcI6bgpO94h5CCnlj8xFKPq7qGixdwGUA6b_ZI3
      c4cZ8eu73VMNrrn_z3WTZlExlpT9XVj-
      ivhfJ4a6T20EtMM5qwIDAQABo4GsMIGpMHQGA1UdIwRtMGuhXqRcMFoxCzAJBgNVBA
      YTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRMwEQYDVQQHDApDYWxpZm9ybmlhMSEw
      HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGSCCQCX9nxZBNQWDjAJBgNVHR
      MEAjAAMA4GA1UdDwEB_wQEAwIGwDAWBgNVHSUBAf8EDDAKBggrBgEFBQcDAzANBgkq
      hkiG9w0BAQUFAAOBgQAGkz9QpoxghZUWT4ivem4cIckfxzTBBiPHCjrrjB2X8Ktn8G
      SZ1MdyIZV8fwdEmD90IvtMHgtzK-
      9wo6Aibj_rVIpxGb7trP82uzc2X8VwYnQbuqQyzofQvcwZHLYplvi95pZ5fVrJvnYA
      UBFyfrdT5GjqL1nqH3a_Y3QPscuCjg"
    },
    "signature":"
    Twajmt_BBLIMcNrDsjqr8lI7O7lEQxXZNhlUOtFkOMMqf37wOPKtp_99LoS82CVmdpCo
    PLaws8zzh-SNIQ42-
    9GYO8_9BaEGCiCwyl8YgWP9fWNfNv2gR2fl2DK4uknkYu1EMBW4YfP81n_pGpb4Gm-
    nMkl4grVZygwAPej3ZZk"
  }
}
```

A.2.3. Sample DeleteTA

A.2.3.1. Sample DeleteTARequest


```
{
  "DeleteTATBSRequest": {
    "ver": "1.0",
    "rid": "req-2",
    "tid": "tran-01",
    "tee": "SecuriTEE",
    "nextdsi": "false",
    "dsihash": "gwjul_9MZks3pqUSN1-eL1aViwGXNAxk0AIKW79dn4U",
    "content": {
      "tsmid": "tsm1.acme.com",
      "sdname": "sd.bank.com",
      "taid": "sd.bank.com.ta"
    }
  }
}
```

```
{
  "DeleteTARequest": {
    "payload":
    "
    eyJEZWxldGVUQVRCU1JlcXVlc3QiOnsidmVyIjoiMS4wIiwicmlkIjoicmVxLTIiLCJ0
    aWQiOiJ0cmFuLTAxIiwidGVlIjoiU2VjdXJpVEVFIiwibmV4dGRzaSI6ImZhbHNlIiwi
    ZHNpaGFzaCI6Imd3anVsXzlNWmtzM3BxVVNOMS1lTDFhVml3R1hOQXhrMEFJS1c3OWRu
    NFUiLCJjb250ZW50Ijp7InByb3RlY3RlZCI6eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0s
    InJlY2lwaWVudHMiOlt7ImhlYWRlciI6eyJhbGciOiJSU0ExXzUifSwiZW5jcnlwdGVk
    X2tleSI6ImtyaGs0d2dpY0RlX3d0VXQyTW4tSUJsdUtvX0JkeXpNY2p1cVlBenBPYnRS
    TG9MZzQ0QkFLN2tRVWE1YTg0TEVJRGEzaHNtWDIxdldNZFJLczN4MTJsOUh5VFdfLUNS
    WmZtcUx2bEh1LV9MSVdvc1ZyRTZVMlJqUnRndllVOWliUkVLczkzRDRHWm4xVHFuZG9n
    d0tXRF9jdG1nWG1sbzZZVXpCWDZhR1dZMCJ9XSwiaXYiOiJBeFk4REN0RGFHbHNiR2xq
    YjNSb1pRIiwiY2lwaGVydGV4dCI6IkhhNzBVdFlUS1ZCa1dEUm4yLTBIX1BGa19yQnpQ
    dGJHdzhSNktlMXotdklNeFBSY0Nxa1puZmwyTjRjUTZPSTZCSHZJUUFoM2Jic0l0dHlR
    bXhDTE5Nbm8wejBrYm9TdkIyVXlxWExpeGVZIiwidGFnIjoidEtUbFRLdlR2LTRtVVlG
    Y1dYWnZMMVlhQnRGNloxVlNxOTMzVmI2UEpmcyJ9fX0",
    "protected" : "eyJhbGciOiJSUzI1NiJ9",
    "header":   {
      "kid":"e9bc097a-ce51-4036-9562-d2ade882db0d",
      "signer":"
      MIIC3zCCAkigAwIBAgIJAJf2fFkE1BYOMA0GCSqGSIb3DQEBBQUAMFoxCzAJBgNVBA
      YTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRMwEQYDVQQHDApDYWxpZm9ybmlhMSEw
      HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQwHhcNMTUwNzAyMDkwMTE4Wh
      cNMjAwNjMwMDkwMTE4WjBaMQswCQYDVQQGEwJVUzETMBEGA1UECAwKQ2FsaWZvcm5p
      YTETMBEGA1UEBwwKQ2FsaWZvcm5pYTEhMB8GA1UECgwYSW50ZXJuZXQgV2lkZ2l0cy
      BQdHkgTHRkMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC8ZtxM1bYickpgSVG-
      meHInI3f_chlMBdL8l7daOEztSs_a6GLqmvSu-
      AoDpTsfEd4EazdMBp5fmgLRGdCYMcI6bgpO94h5CCnlj8xFKPq7qGixdwGUA6b_ZI3
      c4cZ8eu73VMNrrn_z3WTZlExlpT9XVj-
      ivhfJ4a6T20EtMM5qwIDAQABo4GsMIGpMHQGA1UdIwRtMGuhXqRcMFoxCzAJBgNVBA
      YTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRMwEQYDVQQHDApDYWxpZm9ybmlhMSEw
      HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGSCCQCX9nxZBNQWDjAJBgNVHR
      MEAjAAMA4GA1UdDwEB_wQEAwIGwDAWBgNVHSUBAf8EDDAKBggrBgEFBQcDAzANBgkq
      hkiG9w0BAQUFAAOBgQAGkz9QpoxghZUWT4ivem4cIckfxzTBBiPHCjrrjB2X8Ktn8G
      SZ1MdyIZV8fwdEmD90IvtMHgtzK-
      9wo6Aibj_rVIpxGb7trP82uzc2X8VwYnQbuqQyzofQvcwZHLYplvi95pZ5fVrJvnYA
      UBFyfrdT5GjqL1nqH3a_Y3QPscuCjg"
    },
    "signature" :
    "
    BZS0_Ab6pqvGNXe5lqT4Sc3jakyWQeiK9KlVSnimwWnjCCyMtyB9bwvlbILZba3IJiFe
    _3F9bIQpSytGS0f2TQrPTKC7pSjwDw-3kH7HkHcPPJd-
    PpMMfQvRx7AIV8vBqO9MijIC62iN0V2se5z2v8VFjGSoRGgq225w7FvrnWE"
  }
}
```

A.2.3.2. Sample DeleteTAResponse


```
{
  "DeleteTATBSResponse": {
    "ver": "1.0",
    "status": "pass",
        "rid": "req-2",
        "tid": "tran-01",
        "content": {
      "did": "zAHkb0-SQh9U_OT8mR5dB-tygcqpUJ9_x07pIiw8WoM"
    }
  }
}
```

```
{
  "DeleteTAResponse":{
    "payload":"
    ew0KCSJEZWxldGVUQVRCU1Jlc3BvbnNlIjogew0KCQkidmVyIjogIjEuMCIsDQoJCSJz
    dGF0dXMiOiAicGFzcyIsDQoJCSJyaWQiOiAicmVxLTIiLA0KCQkidGlkIjogInRyYW4t
    MDEiLA0KCQkiY29udGVudCI6IHsNCgkJCSJwcm90ZWN0ZWQiOnsiZW5jIjoiQTEyOENC
    Qy1IUzI1NiJ9LA0KCQkJInJlY2lwaWVudHMiOlsNCgkJCQl7DQoJCQkJCSJoZWFkZXIi
    OnsiYWxnIjoiUlNBMV81In0sDQoJCQkJCSJlbmNyeXB0ZWRfa2V5IjoiTXdtU1ZHaWU2
    eHpfQmxTaFlmTFRKRHhKT3oyNWhvYy1HZ2NEM2o5OWFyM2E4X2lYY182ZE44bFRTb1dD
    X19wZEFhaEMyWk5SakdIcTBCZ2JDYTRKalk0eXRkMVBVWDB6M1psbXl1YnRXM291eEpY
    el9PMzg1WGM4S3hySndjbElyZGx2WUY2OVZmeERLQkVzUHJCdzlVenVIa1VmSU4xWlFU
    bWZ0QmVaSlJnIg0KCQkJCX0NCgkJCV0sDQoJCQkiaXYiOiJBeFk4REN0RGFHbHNiR2xq
    YjNSb1pRIiwNCgkJCSJjaXBoZXJ0ZXh0IjoiamhQTlV5ZkFTel9rVV9GbEM2LUtCME01
    WDBHNE5MbHc0LWt0bERyajZTWlUteUp6eUFUbC1oY0ZBWWMwLXJMVEF4cF93N1d1WER0
    Y3N3SzJSSzRjcWciLA0KCQkJInRhZyI6IlBBeGo5N25oT29qVTNIREhxSll4MGZMNWpt
    b0xkTlJkTHRtSmIzUTdrYXciDQoJCX0NCgl9DQp9",
    "protected": "eyJhbGciOiJSUzI1NiJ9",
    "header": {
      "kid":"e9bc097a-ce51-4036-9562-d2ade882db0d",
      "signer":"
      MIIC3zCCAkigAwIBAgIJAJf2fFkE1BYOMA0GCSqGSIb3DQEBBQUAMFoxCzAJ
      BgNVBAYTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRMwEQYDVQQHDApDYWxp
      Zm9ybmlhMSEwHwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQwHhcN
      MTUwNzAyMDkwMTE4WhcNMjAwNjMwMDkwMTE4WjBaMQswCQYDVQQGEwJVUzET
      MBEGA1UECAwKQ2FsaWZvcm5pYTETMBEGA1UEBwwKQ2FsaWZvcm5pYTEhMB8G
      A1UECgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIGfMA0GCSqGSIb3DQEB
      AQUAA4GNADCBiQKBgQC8ZtxM1bYickpgSVG-
      meHInI3f_chlMBdL8l7daOEztSs_a6GLqmvSu-
      AoDpTsfEd4EazdMBp5fmgLRGdCYMcI6bgpO94h5CCnlj8xFKPq7qGixdwGUA
      6b_ZI3c4cZ8eu73VMNrrn_z3WTZlExlpT9XVj-
      ivhfJ4a6T20EtMM5qwIDAQABo4GsMIGpMHQGA1UdIwRtMGuhXqRcMFoxCzAJ
      BgNVBAYTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRMwEQYDVQQHDApDYWxp
      Zm9ybmlhMSEwHwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGSCCQCX
      9nxZBNQWDjAJBgNVHRMEAjAAMA4GA1UdDwEB_wQEAwIGwDAWBgNVHSUBAf8E
      DDAKBggrBgEFBQcDAzANBgkqhkiG9w0BAQUFAAOBgQAGkz9QpoxghZUWT4iv
      em4cIckfxzTBBiPHCjrrjB2X8Ktn8GSZ1MdyIZV8fwdEmD90IvtMHgtzK-
      9wo6Aibj_rVIpxGb7trP82uzc2X8VwYnQbuqQyzofQvcwZHLYplvi95pZ5fV
      rJvnYAUBFyfrdT5GjqL1nqH3a_Y3QPscuCjg"
    },
    "signature":"
    DfoBOetNelKsnAe_m4Z9K5UbihgWNYZsp5jVybiI05sOagDzv6R4do9npaAlAvpNK8HJ
    CxD6D22J8GDUExlIhSR1aDuDCQm6QzmjdkFdxAz5TRYl6zpPCZqgSToN_g1TZxqxEv6V
    Ob5fies4g6MHvCH-Il_-KbHq5YpwGxEEFdg"
  }
}
```

Authors' Addresses

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

Email: mingliang_pei@symantec.com

Nick Cook
Intercede
St. Mary's Road, Lutterworth
Leicestershire, LE17 4PS
Great Britain

Email: nick.cook@intercede.com

Minho Yoo
Solacia
5F, Daerung Post Tower 2, 306 Digital-ro
Seoul 152-790
Korea

Email: paromix@sola-cia.com

Andrew Atyeo
Intercede
St. Mary's Road, Lutterworth
Leicestershire, LE17 4PS
Great Britain

Email: andrew.atyeo@intercede.com

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Rd
Cambridge, CB1 9NJ
Great Britain

Email: Hannes.tschofenig@arm.com