TEEP
Internet-Draft
Intended status: Informational
Expires: June 8, 2020

M. Pei
Symantec
H. Tschofenig
Arm Limited
D. Wheeler
Intel
A. Atyeo
Intercede
L. Dapeng
Alibaba Group
December 06, 2019

Trusted Execution Environment Provisioning (TEEP) Architecture
draft-ietf-teep-architecture-04

Abstract

A Trusted Execution Environment (TEE) is an environment that enforces that only authorized code can execute with that environment, and that any data used by such code cannot be read or tampered with by any code outside that environment. This architecture document motivates the design and standardization of a protocol for managing the lifecycle of trusted applications running inside a TEE.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Applications executing in a device are exposed to many different attacks intended to compromise the execution of the application, or reveal the data upon which those applications are operating. These attacks increase with the number of other applications on the device, with such other applications coming from potentially untrustworthy sources. The potential for attacks further increase with the complexity of features and applications on devices, and the unintended interactions among those features and applications. The danger of attacks on a system increases as the sensitivity of the applications or data on the device increases. As an example, exposure of emails from a mail client is likely to be of concern to its owner, but a compromise of a banking application raises even greater concerns.

The Trusted Execution Environment (TEE) concept is designed to execute applications in a protected environment that enforces that only authorized code can execute with that environment, and that any data used by such code cannot be read or tampered with by any code outside that environment, including a commodity operating system (if present).

This separation reduces the possibility of a successful attack on application components and the data contained inside the TEE. Typically, application components are chosen to execute inside a TEE because those application components perform security sensitive operations or operate on sensitive data. An application component

running inside a TEE is referred to as a Trusted Application (TA), while an application running outside any TEE is referred to as an Untrusted Application (UA).

The TEE typically uses hardware to enforce protections on the TA and its data, but also presents a more limited set of services to applications inside the TEE than is normally available to Untrusted Applications.

But not all TEEs are the same, and different vendors may have different implementations of TEEs with different security properties, different features, and different control mechanisms to operate on TAs. Some vendors may themselves market multiple different TEEs with different properties attuned to different markets. A device vendor may integrate one or more TEEs into their devices depending on market needs.

To simplify the life of developers and service providers interacting with TAs in a TEE, an interoperable protocol for managing TAs running in different TEEs of various devices is needed. In this TEE ecosystem, there often arises a need for an external trusted party to verify the identity, claims, and rights of Service Providers (SP), devices, and their TEEs. This trusted third party is the Trusted Application Manager (TAM).

The Trusted Execution Provisioning (TEEP) protocol addresses the following problems:

- A Service Provider (SP) intending to provide services through a TA to users of a device needs to determine security-relevant information of a device before provisioning their TA to the TEE within the device. An example is the verification of the type of TEE included in a device.

- A TEE in a device needs to determine whether a Service Provider (SP) that wants to manage a TA in the device is authorized to manage TAs in the TEE, and what TAs the SP is permitted to manage.

- The parties involved in the protocol must be able to attest that a TEE is genuine and capable of providing the security protections required by a particular TA.

- A Service Provider (SP) must be able to determine if a TA exists (is installed) on a device (in the TEE), and if not, install the TA in the TEE.

- A Service Provider (SP) must be able to check whether a TA in a device's TEE is the most up-to-date version, and if not, update the TA in the TEE.

- A Service Provider (SP) must be able to remove a TA in a device's TEE if the SP is no longer offering such services or the services are being revoked from a particular user (or device). For example, if a subscription or contract for a particular service has expired, or a payment by the user has not been completed or has been rescinded.

- A Service Provider (SP) must be able to define the relationship between cooperating TAs under the SP's control, and specify whether the TAs can communicate, share data, and/or share key material.

2. Terminology

The following terms are used:

- Untrusted Application: An application running in a Rich Execution Environment, such as an Android, Windows, or iOS application.

- Trusted Application Manager (TAM): An entity that manages Trusted Applications (TAs) running in different TEEs of various devices.

- Device: A physical piece of hardware that hosts one or more TEEs, often along with a Rich Execution Environment. A Device contains a default list of Trust Anchors that identify entities (e.g., TAMs) that are trusted by the Device. This list is normally set by the Device Manufacturer, and may be governed by the Device's network carrier. The list of Trust Anchors is normally modifiable by the Device's owner or Device Administrator. However the Device manufacturer and network carrier may restrict some modifications, for example, by not allowing the manufacturer or carrier's Trust Anchor to be removed or disabled.

- Rich Execution Environment (REE): An environment that is provided and governed by a typical OS (e.g., Linux, Windows, Android, iOS), potentially in conjunction with other supporting operating systems and hypervisors; it is outside of any TEE. This environment and applications running on it are considered untrusted.

- Service Provider (SP): An entity that wishes to provide a service on Devices that requires the use of one or more Trusted Applications. A Service Provider requires the help of a TAM in order to provision the Trusted Applications to remote devices.

- Device User: A human being that uses a device. Many devices have a single device user. Some devices have a primary device user with other human beings as secondary device users (e.g., parent allowing children to use their tablet or laptop). Other devices are not used by a human being and hence have no device user. Relates to Device Owner and Device Administrator.

- Device Owner: A device is always owned by someone. In some cases, it is common for the (primary) device user to also own the device, making the device user/owner also the device administrator. In enterprise environments it is more common for the enterprise to own the device, and any device user has no or limited administration rights. In this case, the enterprise appoints a device administrator that is not the device owner.

- Device Administrator (DA): An entity that is responsible for administration of a Device, which could be the device owner. A Device Administrator has privileges on the Device to install and remove applications and TAs, approve or reject Trust Anchors, and approve or reject Service Providers, among possibly other privileges on the Device. A Device Administrator can manage the list of allowed TAMs by modifying the list of Trust Anchors on the Device. Although a Device Administrator may have privileges and Device-specific controls to locally administer a device, the Device Administrator may choose to remotely administrate a device through a TAM.

- Trust Anchor: As defined in [RFC6024] and [I-D.ietf-suit-manifest], "A trust anchor represents an authoritative entity via a public key and associated data. The public key is used to verify digital signatures, and the associated data is used to constrain the types of information for which the trust anchor is authoritative." The Trust Anchor may be a certificate or it may be a raw public key along with additional data if necessary such as its public key algorithm and parameters.

- Trust Anchor Store: As defined in [RFC6024], "A trust anchor store is a set of one or more trust anchors stored in a device. A device may have more than one trust anchor store, each of which may be used by one or more applications." As noted in [I-D.ietf-suit-manifest], a trust anchor store must resist modification against unauthorized insertion, deletion, and modification.

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

- Trusted Execution Environment (TEE): An execution environment that enforces that only authorized code can execute within the TEE, and data used by that code cannot be read or tampered with by code outside the TEE. A TEE also generally has a device unique credential that cannot be cloned. There are multiple technologies that can be used to implement a TEE, and the level of security achieved varies accordingly. In addition, TEEs typically use an isolation mechanism between Trusted Applications to ensure that one TA cannot read, modify or delete the data and code of another TA.

3. Use Cases

3.1. Payment

A payment application in a mobile device requires high security and trust about the hosting device. Payments initiated from a mobile device can use a Trusted Application to provide strong identification and proof of transaction.

For a mobile payment application, some biometric identification information could also be stored in a TEE. The mobile payment application can use such information for authentication.

A secure user interface (UI) may be used in a mobile device to prevent malicious software from stealing sensitive user input data. Such an application implementation often relies on a TEE for user input protection.

3.2. Authentication

For better security of authentication, a device may store its sensitive authentication keys inside a TEE, providing TEE-protected security key strength and trusted code execution.

3.3. Internet of Things

The Internet of Things (IoT) has been posing threats to networks and national infrastructures because of existing weak security in devices. It is very desirable that IoT devices can prevent malware from manipulating actuators (e.g., unlocking a door), or stealing or modifying sensitive data such as authentication credentials in the device. A TEE can be the best way to implement such IoT security functions.

TEEs could be used to store variety of sensitive data for IoT devices. For example, a TEE could be used in smart door locks to

store a user's biometric information for identification, and for protecting access the locking mechanism.

3.4. Confidential Cloud Computing

A tenant can store sensitive data in a TEE in a cloud computing server such that only the tenant can access the data, preventing the cloud hosting provider from accessing the data. A tenant can run TAs inside a server TEE for secure operation and enhanced data security. This provides benefits not only to tenants with better data security but also to cloud hosting provider for reduced liability and increased cloud adoption.

4. Architecture

4.1. System Components

The following are the main components in the system. Full descriptions of components not previously defined are provided below. Interactions of all components are further explained in the following paragraphs.

```
+-------------------------------------------+
| Device                                    |
|                          +--------+       |        Service Provider
|    +-------------+       |        |----------+               |
|    | TEE-1       |       | TEEP   |---------+|               |
|    | +--------+  |  +----| Broker |       | ||   +--------+  |
|    | | TEEP   |  |  |    |        |<---+  | |+-->|        |<-+
|    | | Agent  |<----+    |        |    |  | |  +-|  TAM-1 |
|    | +--------+  |       |        |<-+ |  | +->| |        |<-+
|    |             |       +--------+  | |  |    | +--------+  |
|    | +---+ +---+ |                   | |  |    | TAM-2   |    |
|  +-->|TA1| |TA2| |        +-------+  | |  |    +--------+    |
|  | | |   | |   |<---------| App-2 |--+ |  |                  |
|  | | +---+ +---+ |    +-------+    |    |  |    Device Administrator
|  | +-------------+    | App-1 |    |    |  |
|  |                    |       |    |    |  |
|  +--------------------|       |---+    |  |
|                       |       |--------+  |
|                       +-------+           |
+-------------------------------------------+
```

Figure 1: Notional Architecture of TEEP

- Service Providers (SP) and Device Administrators (DA) utilize the services of a TAM to manage TAs on Devices. SPs do not directly

interact with devices. DAs may elect to use a TAM for remote administration of TAs instead of managing each device directly.

- Trusted Application Manager (TAM): A TAM is responsible for performing lifecycle management activity on TA's on behalf of Service Providers and Device Administrators. This includes creation and deletion of TA's, and may include, for example, over-the-air updates to keep an SP's TAs up-to-date and clean up when a version should be removed. TAMs may provide services that make it easier for SPs or DAs to use the TAM's service to manage multiple devices, although that is not required of a TAM.

  The TAM performs its management of TA's through an interaction with a Device's TEEP Broker. As shown in Figure 1, the TAM cannot directly contact a Device, but must wait for the TEEP Broker to contact the TAM requesting a particular service. This architecture is intentional in order to accommodate network and application firewalls that normally protect user and enterprise devices from arbitrary connections from external network entities.

  A TAM may be publicly available for use by many SPs, or a TAM may be private, and accessible by only one or a limited number of SPs. It is expected that manufacturers and carriers will run their own private TAM. Another example of a private TAM is a TAM running as a Software-as-a-Service (SaaS) within an SP.

  A SP or Device Administrator chooses a particular TAM based on whether the TAM is trusted by a Device or set of Devices. The TAM is trusted by a device if the TAM's public key is an authorized Trust Anchor in the Device. A SP or Device Administrator may run their own TAM, however the Devices they wish to manage must include this TAM's pubic key in the Trust Anchor list.

  A SP or Device Administrator is free to utilize multiple TAMs. This may be required for a SP to manage multiple different types of devices from different manufacturers, or devices on different carriers, since the Trust Anchor list on these different devices may contain different TAMs. A Device Administrator may be able to add their own TAM's public key or certificate to the Trust Anchor list on all their devices, overcoming this limitation.

  Any entity is free to operate a TAM. For a TAM to be successful, it must have its public key or certificate installed in Devices Trust Anchor list. A TAM may set up a relationship with device manufacturers or carriers to have them install the TAM's keys in their device's Trust Anchor list. Alternatively, a TAM may publish its certificate and allow Device Administrators to install the TAM's certificate in their devices as an after-market-action.

- TEEP Broker: The TEEP Broker is an application component running in a Rich Execution Environment (REE) that enables the message protocol exchange between a TAM and a TEE in a device. The TEEP Broker does not process messages on behalf of a TEE, but merely is responsible for relaying messages from the TAM to the TEE, and for returning the TEE's responses to the TAM.

- TEEP Agent: the TEEP Agent is a processing module running inside a TEE that receives TAM requests that are relayed via a TEEP Broker that runs in an REE. A TEEP Agent in the TEE may parse requests or forward requests to other processing modules in a TEE, which is up to a TEE provider's implementation. A response message corresponding to a TAM request is sent by a TEEP Agent back to a TEEP Broker.

- Certification Authority (CA): Certificate-based credentials used for authenticating a device, a TAM and an SP. A device embeds a list of root certificates (Trust Anchors), from trusted CAs that a TAM will be validated against. A TAM will remotely attest a device by checking whether a device comes with a certificate from a CA that the TAM trusts. The CAs do not need to be the same; different CAs can be chosen by each TAM, and different device CAs can be used by different device manufacturers.

4.2. Different Renditions of TEEP Architecture

There is nothing prohibiting a device from implementing multiple TEEs. In addition, some TEEs (for example, SGX) present themselves as separate containers within memory without a controlling manager within the TEE. In these cases, the Rich Execution Environment hosts multiple TEEP brokers, where each Broker manages a particular TEE or set of TEEs. Enumeration and access to the appropriate TEEP Broker is up to the Rich Execution Environment and the Untrusted Applications. Verification that the correct TA has been reached then becomes a matter of properly verifying TA attestations, which are unforgeable. The multiple TEE approach is shown in the diagram below. For brevity, TEEP Broker 2 is shown interacting with only one TAM and UA, but no such limitation is intended to be implied in the architecture.

```
   +-------------------------------------------+
   | Device                                    |
   |                          +--------+       |        Service Provider
   |                          |        |----------+               |
   |    +-------------+       | TEEP   |---------+|               |
   |    | TEE-1       |   +---| Broker |       | ||   +--------+  |
   |    |             |   |   | 1      |<---+  | |+-->|        |<-+
   |    | +-------+   |   |   |        |    |  | |    |        |
   |    | | TEEP  |   |   |   |        |    |  | |    |        |
   |    | | Agent |<------+   |        |    |  | |    |        |
   |    | | 1     |   |       |        |    |  | |    |        |
   |    | +-------+   |       |        |    |  | |    |        |
   |    |             |       |        |    |  | |    |        |
   |    | +---+ +---+ |       |        |    |  | |  +-|  TAM-1 |
   |    | |TA1| |TA2| |       |        |<-+ |  | +->| |        |<-+
   |  +-->|   | |   |<---+    +--------+  | |  |    | +--------+  |
   |  | | +---+ +---+ |  |                | |  |    | TAM-2  |    |
   |  | |             |  |     +-------+  | |  |    +--------+    |
   |  | +-------------+  +-----| App-2 |--+ |  |       ^          |
   |  |                   +-------+   |    |  |       |        Device
   |  +--------------------| App-1 |   |    |  |       |   Administrator
   |                +------|       |   |    |  |       |
   |    +-----------|-+    |       |---+    |  |       |
   |    | TEE-2     | |    |       |--------+  |       |
   |    | +------+  | |    |       |------+    |       |
   |    | | TEEP |  | |    +-------+      |    |       |
   |    | | Agent|<-----+                 |    |       |
   |    | | 2    |  | | |                 |    |       |
   |    | +------+  | | |                 |    |       |
   |    |           | | |                 |    |       |
   |    | +---+     | | |                 |    |       |
   |    | |TA3|<----+ | |  +----------+   |    |       |
   |    | |   |       | |  | TEEP     |<--+    |       |
   |    | +---+       | +--| Broker   |----------------+
   |    |             |    | 2        |        |
   |    +-------------+    +----------+        |
   |                                           |
   +-------------------------------------------+
```

Figure 2: Notional Architecture of TEEP with multiple TEEs

In the diagram above, TEEP Broker 1 controls interactions with the TA's in TEE-1, and TEEP Broker 2 controls interactions with the TA's in TEE-2. This presents some challenges for a TAM in completely managing the device, since a TAM may not interact with all the TEEP Brokers on a particular platform. In addition, since TEE's may be physically separated, with wholly different resources, there may be no need for TEEP Brokers to share information on installed TAs or

resource usage. However, the architecture guarantees that the TAM will receive all the relevant information from the TEEP Broker to which it communicates.

4.3. Multiple TAMs and Relationship to TAs

As shown in Figure 2, the TEEP Broker provides connections from the TEE and the Untrusted Application to one or more TAMs. The selection of which TAM to communicate with is dependent on information from the Untrusted Application and is directly related to the TA.

When a SP offers a service which requires a TA, the SP associates that service with a specific TA. The TA itself is digitally signed, protecting its integrity, but the signature also links the TA back to the signer. The signer is usually the SP, but in some cases may be another party that the SP trusts. The SP selects one or more TAMs through which to offer their service, and communicates the information of the service and the specific Untrusted Applications and TAs to the TAM.

The SP chooses TAMs based upon the markets into which the TAM can provide access. There may be TAMs that provide services to specific types of mobile devices, or mobile device operating systems, or specific geographical regions or network carriers. A SP may be motivated to utilize multiple TAMs for its service in order to maximize market penetration and availability on multiple types of devices. This likely means that the same service will be available through multiple TAMs.

When the SP publishes the Untrusted Application to an app store or other app repositories, the SP binds the Untrusted Application with a manifest that identifies what TAMs can be contacted for the TA. In some situations, an SP may use only a single TAM - this is likely the case for enterprise applications or SPs serving a closed community. For broad public apps, there will likely be multiple TAMs in the manifest - one servicing one brand of mobile device and another servicing a different manufacturer, etc. Because different devices and different manufacturers trust different TAMs, the manifest will include different TAMs that support this SP's Untrusted Application and TA. Multiple TAMs allow the SP to provide their service and this app (and TA) to multiple different devices.

When a TEEP Broker receives a request from an Untrusted Application to install a TA, a list of TAM URIs may be provided for that TA, and the request is passed to the TEEP Agent. If the TEEP Agent decides that the TA needs to be installed, the TEEP Agent selects a single TAM URI that is consistent with the list of trusted TAMs provisioned on the device invokes the HTTP transport for TEEP to connect to the

TAM URI and begins a TEEP protocol exchange. When the TEEP Agent subsequently receives the TA to install and the TA's manifest indicates dependencies on any other trusted components, each dependency can include a list of TAM URIs for the relevant dependency. If such dependencies exist that are prerequisites to install the TA, then the TEEP Agent recursively follows the same procedure for each dependency that needs to be installed or updated, including selecting a TAM URI that is consistent with the list of trusted TAMs provisioned on the device, and beginning a TEEP exchange. If multiple TAM URIs are considered trusted, only one needs to be contacted and they can be attempted in some order until one responds.

Separate from the Untrusted Application's manifest, this framework relies on the use of the manifest format in [I-D.ietf-suit-manifest] for expressing how to install the TA as well as dependencies on other TEE components and versions. That is, dependencies from TAs on other TEE components can be expressed in a SUIT manifest, including dependencies on any other TAs, or trusted OS code (if any), or trusted firmware. Installation steps can also be expressed in a SUIT manifest.

For example, TEE's compliant with Global Platform may have a notion of a "security domain" (which is a grouping of one or more TAs installed on a device, that can share information within such a group) that must be created and into which one or more TAs can then be installed. It is thus up to the SUIT manifest to express a dependency on having such a security domain existing or being created first, as appropriate.

Updating a TA may cause compatibility issues with any Untrusted Applications or other components that depend on the updated TA, just like updating the OS or a shared library could impact an Untrusted Application. Thus, an implementation needs to take into account such issues.

4.4. Untrusted Apps, Trusted Apps, and Personalization Data

In TEEP, there is an explicit relationship and dependence between the Untrusted Application in the REE and one or more TAs in the TEE, as shown in Figure 2. For most purposes, an Untrusted Application that uses one or more TA's in a TEE appears no different from any other Untrusted Application in the REE. However, the way the Untrusted Application and its corresponding TA's are packaged, delivered, and installed on the device can vary. The variations depend on whether the Untrusted Application and TA are bundled together or are provided separately, and this has implications to the management of the TAs in the TEE. In addition to the Untrusted Application and TA, the TA

and/or TEE may require some additional data to personalize the TA to the service provider or the device or a user. This personalization data is dependent on the TEE, the TA and the SP; an example of personalization data might be username and password of an account with the SP, or a secret symmetric key used by the TA to communicate with the SP. The personalization data must be encrypted to preserve the confidentiality of potentially sensitive data contained within it. Other than this requirement to support confidentiality, TEEP place no limitations or requirements on the personalization data.

There are three possible cases for bundling of the Untrusted Application, TA, and personalization data:

1. The Untrusted Application, TA, and personalization data are all bundled together in a single package by the SP and provided to the TEEP Broker through the TAM.

2. The Untrusted Application and the TA are bundled together in a single package, which a TAM or a publicly accessible app store maintains, and the personalization data is separately provided by the SP's TAM.

3. All components are independent. The Untrusted Application is installed through some independent or device-specific mechanism, and the TAM provides the TA and personalization data from the SP. Delivery of the TA and personalization data may be combined or separate.

The TEEP protocol treats the TA, any dependencies the TA has, and personalization data as separate components with separate installation steps that are expressed in SUIT manifests, and a SUIT manifest might contain or reference multiple binaries (see {{I-D.ietf-suit-manifest} for more details). The TEEP Agent is responsible for handling any installation steps that need to be performed inside the TEE, such as decryption of private TA bianries or personalization data.

4.5. Examples of Application Delivery Mechanisms in Existing TEEs

In order to better understand these cases, it is helpful to review actual implementations of TEEs and their application delivery mechanisms.

In Intel Software Guard Extensions (SGX), the Untrusted Application and TA are typically bundled into the same package (Case 2). The TA exists in the package as a shared library (.so or .dll). The Untrusted Application loads the TA into an SGX enclave when the Untrusted Application needs the TA. This organization makes it easy

to maintain compatibility between the Untrusted Application and the TA, since they are updated together. It is entirely possible to create an Untrusted Application that loads an external TA into an SGX enclave and use that TA (Case 3). In this case, the Untrusted Application would require a reference to an external file or download such a file dynamically, place the contents of the file into memory, and load that as a TA. Obviously, such file or downloaded content must be properly formatted and signed for it to be accepted by the SGX TEE. In SGX, for Case 2 and Case 3, the personalization data is normally loaded into the SGX enclave (the TA) after the TA has started. Although Case 1 is possible with SGX, there are no instances of this known to be in use at this time, since such a construction would require a special installation program and SGX TA to receive the encrypted binary, decrypt it, separate it into the three different elements, and then install all three. This installation is complex, because the Untrusted Application decrypted inside the TEE must be passed out of the TEE to an installer in the REE which would install the Untrusted Application; this assumes that the Untrusted Application package includes the TA code also, since otherwise there is a significant problem in getting the SGX enclave code (the TA) from the TEE, through the installer and into the Untrusted Application in a trusted fashion. Finally, the personalization data would need to be sent out of the TEE (encrypted in an SGX encalve-to-enclave manner) to the REE's installation app, which would pass this data to the installed Untrusted Application, which would in turn send this data to the SGX enclave (TA). This complexity is due to the fact that each SGX enclave is separate and does not have direct communication to other SGX enclaves.

In ARM TrustZone based environments, the Untrusted Application and TA may or may not be bundled together. This differs from SGX since in TrustZone the TA lifetime is not inherently tied to a specific Untrused Application process lifetime as occurs in SGX. A TA is loaded by a trusted OS running in the TEE, where the trusted OS is separate from the OS in the REE. Thus Cases 2 and 3 are equally applicable. In addition, it is possible for TAs to communicate with each other without involving the Untrusted Application, and so the complexity of Case 1 is lower than in the SGX example, and so Case 1 is possible as well though still more complex than Cases 2 and 3.

4.6. Entity Relations

This architecture leverages asymmetric cryptography to authenticate a device to a TAM. Additionally, a TEE in a device authenticates a TAM and TA signer. The provisioning of Trust Anchors to a device may be different from one use case to the other. A device administrator may want to have the capability to control what TAs are allowed. A device manufacturer enables verification of the TA signers and TAM

providers; it may embed a list of default Trust Anchors that the signer of an allowed TA's signer certificate should chain to. A device administrator may choose to accept a subset of the allowed TAs via consent or action of downloading.

```
    (App Developer)    (App Store)    (TAM)     (Device with TEE)  (CAs)
           |                                             |
           |                               --> (Embedded TEE cert) <--
           |                                             |
           | <-----------------------------  Get an app cert ----- |
           |                           | <--  Get a TAM cert ------ |
           |
   1. Build two apps:
      Untrusted Application
          TA
           |
           |
      Untrusted Application -- 2a. --> | ----- 3. Install -------> |
          TA ----------------- 2b. Supply ------> | 4. Messaging-->|
           |                           |          |                |
```

Figure 3: Developer Experience

Figure 3 shows an application developer building two applications: 1) an Untrusted Application; 2) a TA that provides some security functions to be run inside a TEE. At step 2, the application developer uploads the Untrusted Application (2a) to an Application Store. The Untrusted Application may optionally bundle the TA binary. Meanwhile, the application developer may provide its TA to a TAM provider that will be managing the TA in various devices. 3. A user will go to an Application Store to download the Untrusted Application. The Untrusted Application will trigger TA installation by initiating communication with a TAM. This is the step 4. The Untrusted Application will get messages from TAM, and interacts with device TEE via an Agent.

The main components consist of a set of standard messages created by a TAM to deliver TA management commands to a device, and device attestation and response messages created by a TEE that responds to a TAM's message.

It should be noted that network communication capability is generally not available in TAs in today's TEE-powered devices. Trusted Applications need to rely on a broker in the REE to interact with a TEE for network message exchanges. Consequently, a TAM generally communicates with an Untrusted Application about how it gets messages that originate from a TEE inside a device. Similarly, a TA or TEE

generally gets messages from a TAM via a TEEP Broker in this protocol architecture, not directly from the network.

It is imperative to have an interoperable protocol to communicate with different TAMs and different TEEs in different devices. This is the role of the Broker, which is a software component that bridges communication between a TAM and a TEE. Furthermore the Broker communicates with a Agent inside a TEE that is responsible to process TAM requests. The Broker in REE does not need to know the actual content of messages except for the TEE routing information.

5. Keys and Certificate Types

This architecture leverages the following credentials, which allow delivering end-to-end security between a TAM and a TEEP Agent, without relying on any transport security.

Figure 4 summarizes the relationships between various keys and where they are stored. Each public/private key identifies an SP, TAM, or TEE, and gets a certificate that chains up to some CA. A list of trusted certificates is then used to check a presented certificate against.

Different CAs can be used for different types of certificates. TEEP messages are always signed, where the signer key is the message originator's private key such as that of a TAM, or a TEE's private key. In addition to the keys shown in Figure 4, there may be additional keys used for attestation. Refer to the RATS Architecture for more discussion.

Purpose	Cardinality & Location of Private Key	Private Key Signs	Location of Corresponding CA Certs
Authenticating TEE	1 per TEE	TEEP responses	TAM
Authenticating TAM	1 per TAM	TEEP requests	TEEP Agent
Code Signing	1 per SP	TA binary	TEE

Figure 4: Keys

The TEE key pair and certificate are used for authenticating the TEE to a remote TAM. Often, the key pair is burned into the TEE by the TEE manufacturer and the key pair and its certificate are valid for the expected lifetime of the TEE. A TAM provider is responsible for

configuring its TAM with the manufacturer certificates or CAs that are used to sign TEE keys.

The TAM key pair and certificate are used for authenticating a TAM to a remote TEE. A TAM provider is responsible for acquiring a certificate from a CA that is trusted by the TEEs it manages.

The SP key pair and certificate are used to sign TAs that the TEE will consider authorized to execute. TEEs must be configured with the CAs that it considers authorized to sign TAs that it will execute.

5.1. Trust Anchors in TEE

A TEEP Agent's Trust Anchor store contains a list of Trust Anchors, which are CA certificates that sign various TAM certificates. The list is typically preloaded at manufacturing time, and can be updated using the TEEP protocol if the TEE has some form of "Trust Anchor Manager TA" that has Trust Anchors in its configuration data. Thus, Trust Anchors can be updated similar to updating the configuration data for any other TA.

When Trust Anchor update is carried out, it is imperative that any update must maintain integrity where only authentic Trust Anchor list from a device manufacturer or a Device Administrator is accepted. This calls for a complete lifecycle flow in authorizing who can make Trust Anchor update and whether a given Trust Anchor list are non-tampered from the original provider. The signing of a Trust Anchor list for integrity check and update authorization methods are desirable to be developed. This can be addressed outside of this architecture document.

Before a TAM can begin operation in the marketplace to support a device with a particular TEE, it must obtain a TAM certificate from a CA that is listed in the Trust Anchor store of the TEE.

5.2. Trust Anchors in TAM

The Trust Anchor store in a TAM consists of a list of Trust Anchors, which are CA certificates that sign various device TEE certificates. A TAM will accept a device for TA management if the TEE in the device uses a TEE certificate that is chained to a CA that the TAM trusts.

5.3. Scalability

This architecture uses a PKI. Trust Anchors exist on the devices to enable the TEE to authenticate TAMs, and TAMs use Trust Anchors to authenticate TEEs. Since a PKI is used, many intermediate CA

certificates can chain to a root certificate, each of which can issue many certificates. This makes the protocol highly scalable. New factories that produce TEEs can join the ecosystem. In this case, such a factory can get an intermediate CA certificate from one of the existing roots without requiring that TAMs are updated with information about the new device factory. Likewise, new TAMs can join the ecosystem, providing they are issued a TAM certificate that chains to an existing root whereby existing TEEs will be allowed to be personalized by the TAM without requiring changes to the TEE itself. This enables the ecosystem to scale, and avoids the need for centralized databases of all TEEs produced or all TAMs that exist.

5.4. Message Security

Messages created by a TAM are used to deliver TA management commands to a device, and device attestation and messages created by the device TEE to respond to TAM messages.

These messages are signed end-to-end between a TEEP Agent and a TAM, and are typically encrypted such that only the targeted device TEE or TAM is able to decrypt and view the actual content.

6. TEEP Broker

A TEE and TAs often do not have the capability to directly communicate outside of the hosting device. For example, GlobalPlatform [GPTEE] specifies one such architecture. This calls for a software module in the REE world to handle network communication with a TAM.

A TEEP Broker is an application component running in the REE of the device or an SDK that facilitates communication between a TAM and a TEE. It also provides interfaces for Untrusted Applications to query and trigger TA installation that the application needs to use.

An Untrusted Application might communicate with the TEEP Broker at runtime to trigger TA installation itself. Or an Untrusted Application might simply have a metadata file that describes the TAs it depends on and the associated TAM(s) for each TA, and an REE Application Installer can inspect this application metadata file and invoke the TEEP Broker to trigger TA installation on behalf of the Untrusted Application without requiring the Untrusted Application to run first.

6.1. Role of the TEEP Broker

A TEEP Broker abstracts the message exchanges with a TEE in a device. The input data is originated from a TAM or the first initialization call to trigger a TA installation.

The Broker doesn't need to parse a message content received from a TAM that should be processed by a TEE. When a device has more than one TEE, one TEEP Broker per TEE could be present in REE. A TEEP Broker interacts with a TEEP Agent inside a TEE.

A TAM message may indicate the target TEE where a TA should be installed. A compliant TEEP protocol should include a target TEE identifier for a TEEP Broker when multiple TEEs are present.

The Broker relays the response messages generated from a TEEP Agent in a TEE to the TAM. The Broker is not expected to handle any network connection with an application or TAM.

The Broker only needs to return an 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 TAM.

6.2. TEEP Broker Implementation Consideration

A Provider should consider methods of distribution, scope and concurrency on devices and runtime options when implementing a TEEP Broker. 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.

6.2.1. TEEP Broker APIs

The following conceptual APIs exist from a TEEP Broker to a TEEP Agent:

1. RequestTA: A notification from an REE application (e.g., an installer, or a normal application) that it depends on a given TA, which may or may not already be installed in the TEE.

2. ProcessTeepMessage: A message arriving from the network, to be delivered to the TEEP Agent for processing.

3. RequestPolicyCheck: A hint (e.g., based on a timer) that the TEEP Agent may wish to contact the TAM for any changes, without the device itself needing any particular change.

4. ProcessError: A notification that the TEEP Broker could not deliver an outbound TEEP message to a TAM.

For comparison, similar APIs may exist on the TAM side, where a Broker may or may not exist (depending on whether the TAM uses a TEE or not):

1. ProcessConnect: A notification that an incoming TEEP session is being requested by a TEEP Agent.

2. ProcessTeepMessage: A message arriving from the network, to be delivered to the TAM for processing.

For further discussion on these APIs, see [I-D.ietf-teep-otrp-over-http].

6.2.2. TEEP Broker Distribution

The Broker installation is commonly carried out at OEM time. A user can dynamically download and install a Broker on-demand.

6.2.3. Number of TEEP Brokers

There should be generally only one shared TEEP Broker in a device. The device's TEE vendor will most probably supply one Broker. When multiple TEEs are present in a device, one TEEP Broker per TEE may be used.

When only one Broker is used per device, the Broker provider is responsible to allow multiple TAMs and TEE providers to achieve interoperability. With a standard Broker interface, each TAM can implement its own SDK for its SP Untrusted Applications to work with this Broker.

Multiple independent Broker providers can be used as long as they have standard interface to an Untrusted Application or TAM SDK. Only one Broker is generally expected in a device.

7. Attestation

Attestation is the process through which one entity (an Attester) presents "evidence", in the form of a series of claims, to another entity (a Verifier), and provides sufficient proof that the claims are true. Different verifiers may have different standards for attestation proofs and not all attestations are acceptable to every verifier. A third entity (a Relying Party) can then use "attestation results", in the form of another series of claims, from a Verifier to make authorization decisions.

In TEEP, as depicted in Figure 5, the primary purpose of an attestation is to allow a device (the Attester) to prove to TAMs (the Relying Parties) that a TEE in the device has particular properties, was built by a particular manufacturer, or is executing a particular TA. Other claims are possible; TEEP does not limit the claims that may appear in evidence or attestation results, but defines a minimal set of attestation result claims required for TEEP to operate properly. Extensions to these claims are possible. Other standards or groups may define the format and semantics of extended claims.

```
+----------------+
| Device         |            +----------+
| +------------+ |  Evidence  |   TAM    |   Evidence    +----------+
| |     TEE    |------------->| (Relying |-------------->| Verifier |
| | (Attester) | |            |  Party)  |<--------------|          |
| +------------+ |            +----------+  Attestation  +----------+
+----------------+                              Result
```

Figure 5: TEEP Attestation Roles

As of the writing of this specification, device and TEE attestations have not been standardized across the market. Different devices, manufacturers, and TEEs support different attestation algorithms and mechanisms. In order for TEEP to be inclusive, it is agnostic to the format of evidence, allowing proprietary or standardized formats to be used between a TEE and a verifier (which may or may not be colocated in the TAM). However, it should be recognized that not all verifiers may be able to process all proprietary forms of attestation evidence. Similarly, the TEEP protocol is agnostic as to the format of attestation results, and the protocol (if any) used between the TAM and a verifier, as long as they convey at least the required set of claims in some format.

The assumptions which may apply to an attestation have to do with the quality of the attestation and the quality and security provided by the TEE, the device, the manufacturer, or others involved in the device or TEE ecosystem. Some of the assumptions that might apply to an attestations include (this may not be a comprehensive list):

- Assumptions regarding the security measures a manufacturer takes when provisioning keys into devices/TEEs;

- Assumptions regarding what hardware and software components have access to the Attestation keys of the TEE;

- Assumptions related to the source or local verification of claims within an attestation prior to a TEE signing a set of claims;

- Assumptions regarding the level of protection afforded to attestation keys against exfiltration, modification, and side channel attacks;

- Assumptions regarding the limitations of use applied to TEE Attestation keys;

- Assumptions regarding the processes in place to discover or detect TEE breeches; and

- Assumptions regarding the revocation and recovery process of TEE attestation keys.

TAMs must be comfortable with the assumptions that are inherently part of any attestation result they accept. Alternatively, any TAM may choose not to accept an attestation result generated using evidence from a particular manufacturer or device's TEE based on the inherent assumptions. The choice and policy decisions are left up to the particular TAM.

Some TAMs may require additional claims in order to properly authorize a device or TEE. These additional claims may help clear up any assumptions for which the TAM wants to alleviate. The specific format for these additional claims are outside the scope of this specification, but the TEEP protocol allows these additional claims to be included in the attestation messages.

7.1. Information Required in TEEP Claims

- Device Identifying Info: TEEP attestations must uniquely identify a device to the TAM and SP. This identifier allows the TAM to provide services unique to the device, such as managing installed TAs, and providing subscriptions to services, and locating device-specific keying material to communicate with or authenticate the device. Additionally, device manufacturer information must be provided to provide better universal uniqueness qualities without requiring globally unique identifiers for all devices.

- TEE Identifying info: The type of TEE that generated this attestation must be identified. Standard TEE types are identified by an IANA number, but also must include version identification information such as the hardware, firmware, and software version of the TEE, as applicable by the TEE type. TEE manufacturer information for the TEE is required in order to disambiguate the same TEE type created by different manufacturers and resolve potential assumptions around manufacturer provisioning, keying and support for the TEE.

- Liveness Proof: A claim that includes liveness information must be included, such as a nonce or timestamp.

- Requested Components: A list of zero or more components (TAs or other dependencies needed by a TEE) that are requested by some depending app, but which are not currently installed in the TEE.

8. Algorithm and Attestation Agility

RFC 7696 [RFC7696] outlines the requirements to migrate from one mandatory-to-implement algorithm suite to another over time. This feature is also known as crypto agility. Protocol evolution is greatly simplified when crypto agility is already considered during the design of the protocol. In the case of the Trusted Execution Provisioning (TEEP) Protocol the diverse range of use cases, from trusted app updates for smart phones and tablets to updates of code on higher-end IoT devices, creates the need for different mandatory-to-implement algorithms already from the start.

Crypto agility in TEEP concerns the use of symmetric as well as asymmetric algorithms. Symmetric algorithms are used for encryption of content whereas the asymmetric algorithms are mostly used for signing messages.

In addition to the use of cryptographic algorithms in TEEP there is also the need to make use of different attestation technologies. A Device must provide techniques to inform a TAM about the attestation technology it supports. For many deployment cases it is more likely for the TAM to support one or more attestation techniques whereas the Device may only support one.

9. Security Considerations

9.1. 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 TAM 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 protocol.

We allow a way for an Untrusted Application to check the trustworthiness of a TA. A TEEP Broker has a function to allow an application to query the information about a TA.

An Untrusted Application may perform verification of the TA by verifying the signature of the TA. An application can do additional trust checks on the certificate returned for this TA. It might trust the TAM, or require additional SP signer trust chaining.

9.2. One TA Multiple SP Case

A TA for multiple SPs must have a different identifier per SP. They should appear as different TAs when they are installed in the same device.

9.3. Broker Trust Model

A TEEP Broker could be malware in the vulnerable REE. An Untrusted Application will connect its TAM provider for required TA installation. It gets command messages from the TAM, and passes the message to the Broker.

The architecture enables the TAM to communicate with the device's TEE to manage TAs. All TAM messages are signed and sensitive data is encrypted such that the TEEP Broker cannot modify or capture sensitive data.

9.4. Data Protection at TAM and TEE

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

9.5. Compromised CA

A root CA for TAM certificates might get compromised. Some TEE Trust Anchor update mechanism is expected from device OEMs. TEEs are responsible for validating certificate revocation about a TAM certificate chain.

If the root CA of some TEE device certificates is compromised, these devices might be rejected by a TAM, which is a decision of the TAM implementation and policy choice. TAMs are responsible for validating any intermediate CA for TEE device certificates.

9.6. Compromised TAM

Device TEEs are responsible for validating the supplied TAM certificates to determine that the TAM is trustworthy.

9.7. Certificate Renewal

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

9.8. Keeping Secrets from the TAM

In some scenarios, it is desirable to protect the TA binary or configuration from being disclosed to the TAM that distributes them. In such a scenario, the files can be encrypted end-to-end between an SP and a TEE. However, there must be some means of provisioning the decryption key into the TEE and/or some means of the SP securely learning a public key of the TEE that it can use to encrypt. One way to do this is for the SP to run its own TAM, merely to distribute the decryption key via the TEEP protocol, and the key file can be a dependency in the manifest of the encrypted TA. Thus, the TEEP Agent would look at the TA manifest, determine there is a dependency with a TAM URI of the SP's TAM. The Agent would then install the dependency, and then continue with the TA installation steps, including decrypting the TA binary with the relevant key.

10. IANA Considerations

This document does not require actions by IANA.

11. Acknowledgements

Some content of this document is based on text in a previous OTrP protocol document [I-D.ietf-teep-opentrustprotocol]. We thank the former co-authors Nick Cook and Minho Yoo for the initial document content, and contributors Brian Witten, Tyler Kim, and Alin Mutu.

12. Informative References

[GPTEE] Global Platform, "GlobalPlatform Device Technology: TEE System Architecture, v1.1", Global Platform GPD_SPE_009, January 2017, <https://globalplatform.org/specs-library/tee-system-architecture-v1-1/>.

[I-D.ietf-suit-manifest]
Moran, B., Tschofenig, H., and H. Birkholz, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", draft-ietf-suit-manifest-02 (work in progress), November 2019.

[I-D.ietf-teep-opentrustprotocol]
Pei, M., Atyeo, A., Cook, N., Yoo, M., and H. Tschofenig, "The Open Trust Protocol (OTrP)", draft-ietf-teep-opentrustprotocol-03 (work in progress), May 2019.

[I-D.ietf-teep-otrp-over-http]
Thaler, D., "HTTP Transport for Trusted Execution Environment Provisioning: Agent-to- TAM Communication", draft-ietf-teep-otrp-over-http-03 (work in progress), November 2019.

[RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <https://www.rfc-editor.org/info/rfc6024>.

[RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <https://www.rfc-editor.org/info/rfc7696>.

Appendix A. History

RFC EDITOR: PLEASE REMOVE THIS SECTION

IETF Drafts

draft-00: - Initial working group document

Authors' Addresses

Mingliang Pei
Symantec

EMail: mingliang_pei@symantec.com

Hannes Tschofenig
Arm Limited

EMail: hannes.tschofenig@arm.com

David Wheeler
Intel

EMail: david.m.wheeler@intel.com

Andrew Atyeo
Intercede

EMail: andrew.atyeo@intercede.com

Liu Dapeng
Alibaba Group

EMail: maxpassion@gmail.com