Workgroup: TEEP
Internet-Draft:
draft-ietf-teep-architecture-18
Published: 11 July 2022
Intended Status: Informational
Expires: 12 January 2023
Authors: M. Pei (Broadcom) H. Tschofenig (Arm Limited) D. Thaler (Microsoft) D. Wheeler (Amazon)

Trusted Execution Environment Provisioning (TEEP) Architecture

Abstract

A Trusted Execution Environment (TEE) is an environment that enforces that any code within that environment cannot be tampered with, 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 such a TEE.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 12 January 2023.

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 increases 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 let applications execute in a protected environment that enforces that any code within that environment cannot be tampered with, and that any data used by such code cannot be read or tampered with by any code outside that environment, including by a commodity operating system (if present). In a system with multiple TEEs, this also means that code in one TEE cannot be read or tampered with by code in another TEE.

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, i.e., in the Rich Execution Environment (REE), is referred to as an Untrusted Application. In the example of a banking application, code that relates to the authentication protocol could reside in a TA while the application logic including HTTP protocol parsing could be contained in the Untrusted Application. In addition, processing of credit card numbers or account balances could be done in a TA as it is sensitive data. The precise code split is ultimately a decision of the developer based on the assets he or she wants to protect according to the threat model.

TEEs are typically used in cases where software or data assets need to be protected from unauthorised access where threat actors may have physical or administrative access to a device. This situation arises for example in gaming consoles where anti-cheat protection is a concern, devices such as ATMs or IoT devices placed in locations where attackers might have physical access, cell phones or other devices used for mobile payments, and hosted cloud environments.

Such environments can be thought of as hybrid devices where one user or administrator controls the REE and a different (remote) user or administrator controls a TEE in the same physical device. It may also be the case in some constrained devices that there is no REE (only a TEE) and there may be no local "user" per se, only a remote TEE administrator. For further discussion of such confidential computing use cases and threat model, see [CC-Overview] and [CC-Technical-Analysis].

TEEs use hardware enforcement combined with software protection to secure TAs and its data. TEEs typically offer a more limited set of services to TAs than is normally available to Untrusted Applications.

Not all TEEs are the same, however, 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 TA developers interacting with TAs in a TEE, an interoperable protocol for managing TAs running in different TEEs of various devices is needed. This software update protocol needs to make sure that compatible trusted and untrusted components (if any) of an application are installed on the correct device. In this TEE ecosystem, there often arises a need for an external trusted party to verify the identity, claims, and rights of TA developers, devices, and their TEEs. This external trusted party is the Trusted Application Manager (TAM).

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

* An installer of an Untrusted Application that depends on a given TA wants to request installation of that TA in the device's TEE so that the Untrusted Application can complete, but the TEE needs to verify whether such a TA is actually authorized to run in the TEE and consume potentially scarce TEE resources.

* A TA developer providing a TA whose code itself is considered confidential wants to determine security-relevant information of a device before allowing their TA to be provisioned to the TEE within the device. An example is the verification of the type of TEE included in a device and that it is capable of providing the security protections required.

* A TEE in a device wants to determine whether an entity that wants to manage a TA in the device is authorized to manage TAs in the TEE, and what TAs the entity is permitted to manage.

* A Device Administrator wants to determine if a TA exists (is installed) on a device (in the TEE), and if not, install the TA in the TEE.

* A Device Administrator wants 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 Device Administrator wants to remove a TA from a device's TEE if the TA developer is no longer maintaining that TA, when the TA has been revoked, or is not used for other reasons anymore (e.g., due to an expired subscription).

For TEEs that simply verify and load signed TA's from an untrusted filesystem, classic application distribution protocols can be used without modification. The problems in the bullets above, on the other hand, require a new protocol, i.e., the TEEP protocol. The TEEP protocol is a solution for TEEs that can install and enumerate TAs in a TEE-secured location where another domain-specific protocol standard (e.g., [GSMA], [OTRP]) that meets the needs is not already in use.

2. Terminology

The following terms are used:

* App Store: An online location from which Untrusted Applications can be downloaded.

* Device: A physical piece of hardware that hosts one or more TEEs, often along with an REE.

* Device Administrator: 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 Untrusted Applications and TAs, approve or reject Trust Anchors, and approve or reject TA developers, 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 administer a device through a TAM.

* 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 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., a 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.

*Personalization Data: A set of configuration data that is specific to the device or user. The Personalization Data may depend on the type of TEE, a particular TEE instance, the TA, and even the user of the device; an example of Personalization Data might be a secret symmetric key used by a TA to communicate with some service.

*Raw Public Key: A raw public key consists of only the algorithm identifier (type) of the key and the cryptographic public key material, such as the SubjectPublicKeyInfo structure of a PKIX certificate [RFC5280]. Other serialization formats that do not rely on ASN.1 may also be used.

*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 the TEE(s) managed by the TEEP protocol. This environment and applications running on it are considered untrusted (or more precisely, less trusted than a TEE).

*Trust Anchor: As defined in [RFC6024] and [RFC9019], "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, a raw public key or other structure, as appropriate. It can be a non-root certificate when it is a certificate.

*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 [RFC9019], a Trust Anchor Store must resist modification against unauthorized insertion, deletion, and modification.

*Trusted Application (TA): An application (or, in some implementations, an application component) that runs in a TEE.

*Trusted Application Manager (TAM): An entity that manages Trusted Applications and other Trusted Components running in TEEs of various devices.

*Trusted Component: A set of code and/or data in a TEE managed as a unit by a Trusted Application Manager. Trusted Applications and Personalization Data are thus managed by being included in Trusted Components. Trusted OS code or trusted firmware can also be expressed as Trusted Components that a Trusted Component depends on.

*Trusted Component Developer: An entity that develops one or more Trusted Components.

*Trusted Component Signer: An entity that signs a Trusted Component with a key that a TEE will trust. The signer might or might not be the same entity as the Trusted Component Developer. For example, a Trusted Component might be signed (or re-signed) by a Device Administrator if the TEE will only trust the Device Administrator. A Trusted Component might also be encrypted, if the code is considered confidential.

*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.

*Untrusted Application: An application running in an REE. An Untrusted Application might depend on one or more TAs.

3. Use Cases

3.1. Payment

A payment application in a mobile device requires high security and trust in 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 unlocking the device and for local identification of the user.

A trusted user interface (UI) may be used in a mobile device to prevent malicious software from stealing sensitive user input data. Such an implementation often relies on a TEE for providing access to peripherals, such as PIN input or a trusted display, so that the REE cannot observe or tamper with the user input or output.

3.2. Authentication

For better security of authentication, a device may store its keys and cryptographic libraries inside a TEE limiting access to cryptographic functions via a well-defined interface and thereby reducing access to keying material.

3.3. Internet of Things

Weak security in Internet of Things (IoT) devices has been posing threats to critical infrastructure, i.e., assets that are essential for the functioning of a society and economy. It is 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.

3.4. Confidential Cloud Computing

A tenant can store sensitive data, such as customer details or credit card numbers, 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 providers for reduced liability and increased cloud adoption.

4. Architecture

4.1. System Components

Figure 1 shows the main components in a typical device with an REE and a TEE. Full descriptions of components not previously defined are provided below. Interactions of all components are further explained in the following paragraphs.

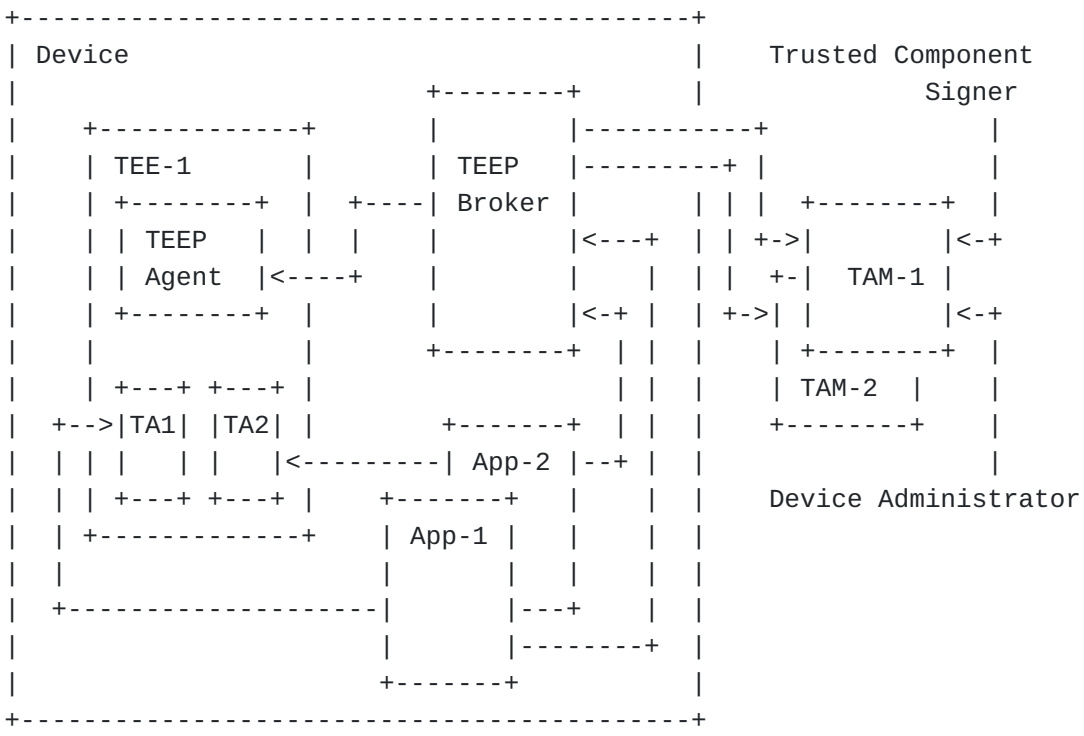

Figure 1: Notional Architecture of TEEP

*Trusted Component Signers and Device Administrators utilize the services of a TAM to manage TAs on devices. Trusted Component Signers do not directly interact with devices. Device Administators 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 Trusted Components on behalf of Trusted Component Signers and Device Administrators. This includes installation and deletion of Trusted Components, and may include, for example, over-the-air updates to keep Trusted Components up-to-date and clean up when Trusted Components should be removed. TAMs may provide services that make it easier for Trusted Component Signers or Device Administators to use the TAM's service to manage multiple devices, although that is not required of a TAM.

The TAM performs its management of Trusted Components on the device through interactions with a device's TEEP Broker, which relays messages between a TAM and a TEEP Agent running inside the TEE. TEEP authentication is performed between a TAM and a TEEP Agent.

As shown in Figure 1, the TAM cannot directly contact a TEEP Agent, 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 Trusted Component Signers, or a TAM may be private, and accessible by only one or a limited number of Trusted Component Signers. It is expected that many enterprises, manufacturers, and network carriers will run their own private TAM.

A Trusted Component Signer 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, or chains up to, an authorized Trust Anchor in the device. A Trusted Component Signer or Device Administrator may run their own TAM, but the devices they wish to manage must include this TAM's public key or certificate, or a certificate it chains up to, in the Trust Anchor Store.

A Trusted Component Signer or Device Administrator is free to utilize multiple TAMs. This may be required for managing Trusted Components on multiple different types of devices from different manufacturers, or mobile devices on different network carriers, since the Trust Anchor Store on these different devices may contain keys for different TAMs. A Device Administrator may be able to add their own TAM's public key or certificate, or a certificate it chains up to, to the Trust Anchor Store 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 a device's Trust Anchor Store. A TAM may set up a relationship with device manufacturers or network carriers to have them install the TAM's keys in their device's Trust Anchor Store. 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: A 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. A 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. In devices with no REE (e.g., a microcontroller where all code runs in an environment that meets the definition of a Trusted Execution Environment in Section 2), the TEEP Broker would be absent and instead the TEEP protocol transport would be implemented inside the TEE itself.

*TEEP Agent: The TEEP Agent is a processing module running inside a TEE that receives TAM requests (typically 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 back to the TAM, again typically relayed via a TEEP Broker.

*Certification Authority (CA): A CA is an entity that issues digital certificates (especially X.509 certificates) and vouches for the binding between the data items in a certificate [RFC4949]. Certificates are then used for authenticating a device, a TAM, or a Trusted Component Signer, as discussed in Section 5. 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. Multiple TEEs in a Device

Some devices might implement multiple TEEs. In these cases, there might be one shared TEEP Broker that interacts with all the TEEs in the device. However, some TEEs (for example, SGX [SGX]) present themselves as separate containers within memory without a controlling manager within the TEE. As such, there might be multiple TEEP Brokers in the REE, where each TEEP Broker communicates with one or more TEEs associated with it.

It is up to the REE and the Untrusted Applications how they select the correct TEEP Broker. Verification that the correct TA has been reached then becomes a matter of properly verifying TA attestations, which are unforgeable.

The multiple TEEP Broker approach is shown in the diagram below. For brevity, TEEP Broker 2 is shown interacting with only one TAM and Untrusted Application and only one TEE, but no such limitations are intended to be implied in the architecture.

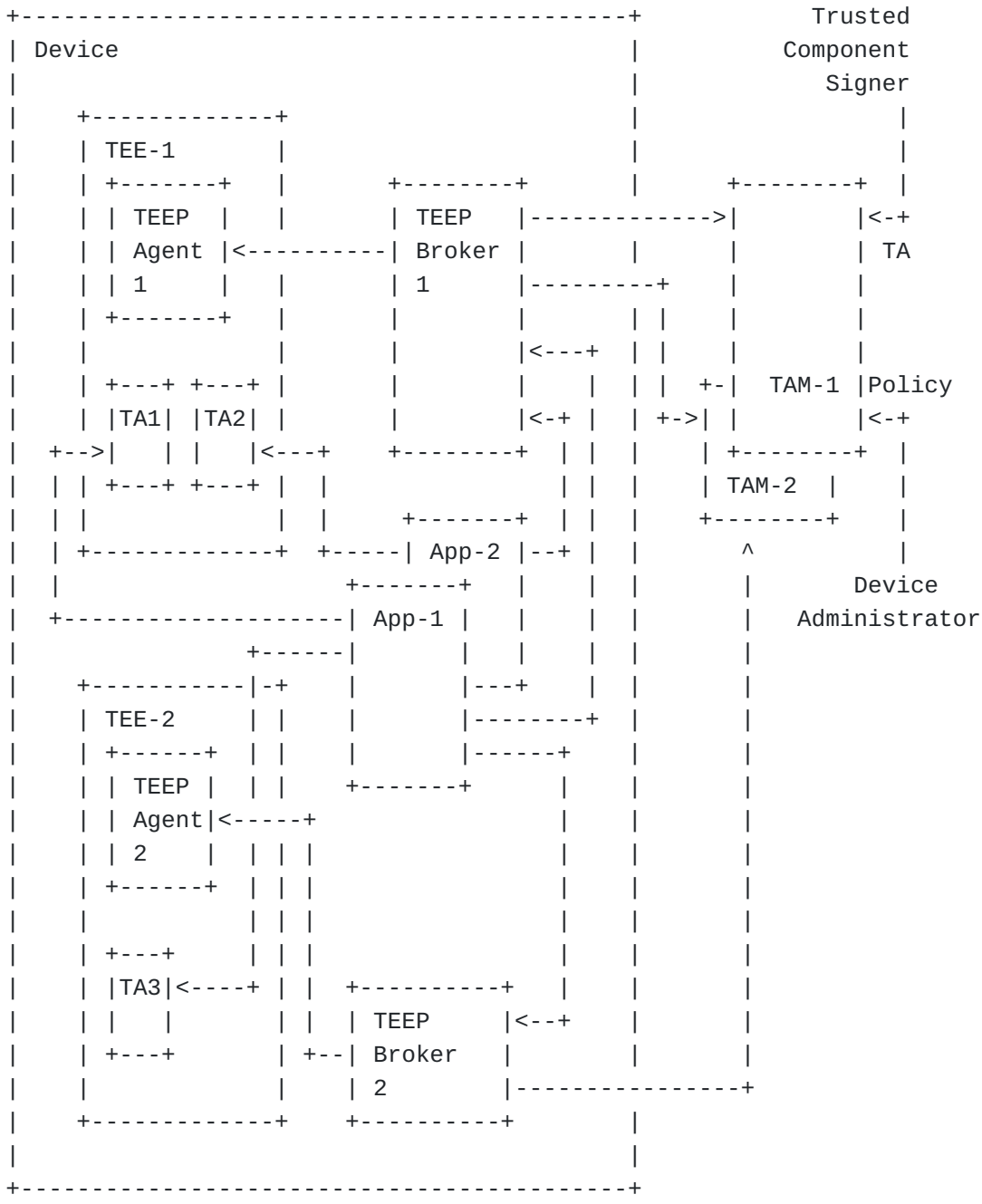

Figure 2: Notional Architecture of TEEP with multiple TEEs

In the diagram above, TEEP Broker 1 controls interactions with the TAs in TEE-1, and TEEP Broker 2 controls interactions with the TAs 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 TEEs may be physically separated, with wholly different resources, there may be no need for TEEP Brokers to share information on installed Trusted Components or resource usage.

4.3. Multiple TAMs and Relationship to TAs

As shown in Figure 2, a TEEP Broker provides communication between one or more TEEP Agents and one or more TAMs. The selection of which TAM to interact with might be made with or without input from an Untrusted Application, but is ultimately the decision of a TEEP Agent.

A TEEP Agent is assumed to be able to determine, for any given Trusted Component, whether that Trusted Component is installed (or minimally, is running) in a TEE with which the TEEP Agent is associated.

Each Trusted Component is digitally signed, protecting its integrity, and linking the Trusted Component back to the Trusted Component Signer. The Trusted Component Signer is often the Trusted Component Developer, but in some cases might be another party such as a Device Administrator or other party to whom the code has been licensed (in which case the same code might be signed by multiple licensees and distributed as if it were different TAs).

A Trusted Component Signer selects one or more TAMs and communicates the Trusted Component(s) to the TAM. For example, the Trusted Component Signer might choose TAMs based upon the markets into which the TAM can provide access. There may be TAMs that provide services to specific types of devices, or device operating systems, or specific geographical regions or network carriers. A Trusted Component Signer may be motivated to utilize multiple TAMs in order to maximize market penetration and availability on multiple types of devices. This means that the same Trusted Component will often be available through multiple TAMs.

When the developer of an Untrusted Application that depends on a Trusted Component publishes the Untrusted Application to an app store or other app repository, the developer optionally binds the Untrusted Application with a manifest that identifies what TAMs can be contacted for the Trusted Component. In some situations, a Trusted Component may only be available via a single TAM - this is likely the case for enterprise applications or Trusted Component Signers serving a closed community. For broad public apps, there will likely be multiple TAMs in the Untrusted Application's 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 can include multiple TAMs that support the required Trusted Component.

When a TEEP Broker receives a request (see the RequestTA API in Section 6.2.1) from an Untrusted Application to install a Trusted Component, a list of TAM URIs may be provided for that Trusted

Component, and the request is passed to the TEEP Agent. If the TEEP Agent decides that the Trusted Component needs to be installed, the TEEP Agent selects a single TAM URI that is consistent with the list of trusted TAMs provisioned in the TEEP Agent, 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 Trusted Component to install and the Trusted Component'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 Trusted Component, 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 a Trusted Component, as well as any dependencies on other TEE components and versions. That is, dependencies from Trusted Components on other Trusted Components can be expressed in a SUIT manifest, including dependencies on any other TAs, trusted OS code (if any), or trusted firmware. Installation steps can also be expressed in a SUIT manifest.

For example, TEEs compliant with GlobalPlatform [GPTEE] 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 Trusted Component may cause compatibility issues with any Untrusted Applications or other components that depend on the updated Trusted Component, 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 an Untrusted Application in an REE and one or more TAs in a TEE, as shown in Figure 2. For most purposes, an Untrusted Application that uses one or more TAs in a TEE appears no different from any other Untrusted Application in the REE. However, the way the Untrusted Application and its corresponding TAs 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 a TEE. In addition to the Untrusted Application and TA(s), the TA(s) and/or TEE may also require additional data to personalize the TA to the device or a user. Implementations must support encryption to preserve the confidentiality of such Personalization Data, which may potentially contain sensitive data. Implementations must also support mechanisms for integrity protection of such Personalization Data. Other than the requirement to support confidentiality and integrity protection, the TEEP architecture places no limitations or requirements on the Personalization Data.

There are multiple possible cases for bundling of an Untrusted Application, TA(s), and Personalization Data. Such cases include (possibly among others):

1. The Untrusted Application, TA(s), and Personalization Data are all bundled together in a single package by a Trusted Component Signer and either provided to the TEEP Broker through the TAM, or provided separately (with encrypted Personalization Data), with key material needed to decrypt and install the Personalization Data and TA provided by a TAM.

2. The Untrusted Application and the TA(s) 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 Personalization Data provider's TAM.

3. All components are independent packages. The Untrusted Application is installed through some independent or device-specific mechanism, and one or more TAMs provide (directly or indirectly by reference) the TA(s) and Personalization Data.

4. The TA(s) and Personalization Data are bundled together into a package provided by a TAM, while the Untrusted Application is installed through some independent or device-specific mechanism such as an app store.

5. Encrypted Personalization Data is bundled into a package distributed with the Untrusted Application, while the TA(s) and key material needed to decrypt and install the Personalization Data are in a separate package provided by a TAM.

The TEEP protocol can treat each TA, any dependencies the TA has, and Personalization Data as separate Trusted 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 binaries or Personalization Data.

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

4.4.1. Example: Application Delivery Mechanisms in Intel SGX

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 (Cases 3-5). 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, any Personalization Data is normally loaded into the SGX enclave (the TA) after the TA has started. Although it is possible with SGX to include the Untrusted Application in an encrypted package along with Personalization Data (Cases 1 and 5), 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 (which might or might not be the TEEP Agent itself based on the implementation) to receive the encrypted package, decrypt it, separate it into the different elements, and then install each one. 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. Finally, the Personalization Data would need to be sent out of the TEE (encrypted in an SGX enclave-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.

As long as signed files (TAs and/or Personalization Data) are installed into an untrusted filesystem and trust is verified by the TEE at load time, classic distribution mechanisms can be used. Some

uses of SGX, however, allow a model where a TA can be dynamically installed into an SGX enclave that provides a runtime platform. The TEEP protocol can be used in such cases, where the runtime platform could include a TEEP Agent.

4.4.2. Example: Application Delivery Mechanisms in Arm TrustZone

In Arm TrustZone [TrustZone] for A-class devices, 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 such as a GlobalPlatform [GPTEE] compliant TEE, where the trusted OS is separate from the OS in the REE. Thus Cases 2-4 are equally applicable. In addition, it is possible for TAs to communicate with each other without involving any Untrusted Application, and so the complexity of Cases 1 and 5 are lower than in the SGX example, though still more complex than Cases 2-4.

A trusted OS running in the TEE (e.g., OP-TEE) that supports loading and verifying signed TAs from an untrusted filesystem can, like SGX, use classic file distribution mechanisms. If secure TA storage is used (e.g., a Replay-Protected Memory Block device) on the other hand, the TEEP protocol can be used to manage such storage.

4.5. Entity Relations

This architecture leverages asymmetric cryptography to authenticate a device to a TAM. Additionally, a TEEP Agent in a device authenticates a TAM. 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 by one or more TAMs and by Trusted Component Signers; it may embed a list of default Trust Anchors into the TEEP Agent and TEE for TAM trust verification and TA signature verification.

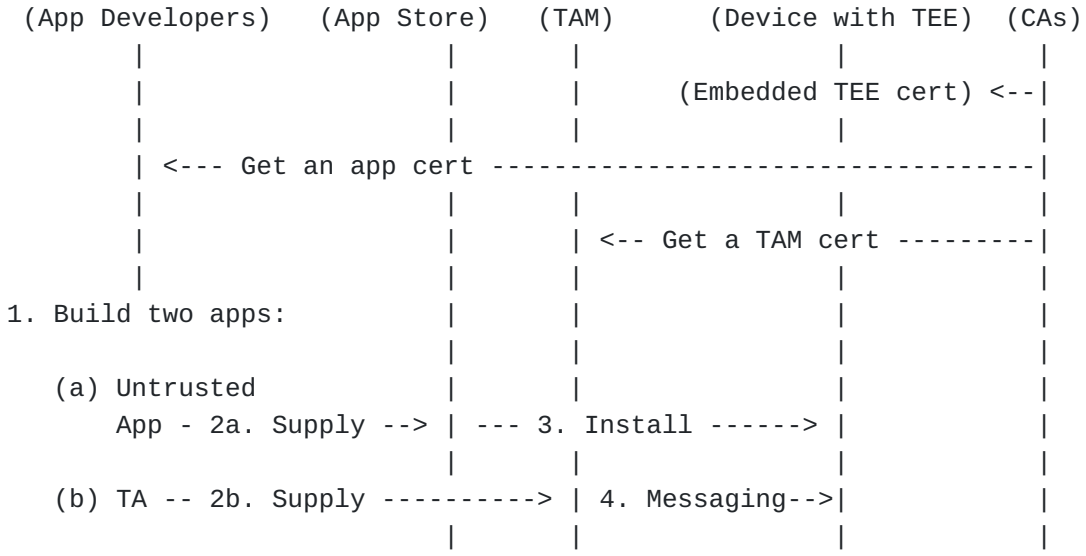

Figure 3: Example Developer Experience

Figure 3 shows an example where the same developer builds and signs two applications: (a) an Untrusted Application; (b) a TA that provides some security functions to be run inside a TEE. This example assumes that the developer, the TEE, and the TAM have previously been provisioned with certificates.

At step 1, the developer authors the two applications.

At step 2, the developer uploads the Untrusted Application (2a) to an Application Store. In this example, the developer is also the Trusted Component Signer, and so generates a signed TA. The developer can then either bundle the signed TA with the Untrusted Application, or the developer can provide a signed Trusted Component containing the TA to a TAM that will be managing the TA in various devices.

At step 3, a user will go to an Application Store to download the Untrusted Application (where the arrow indicates the direction of data transfer).

At step 4, since the Untrusted Application depends on the TA, installing the Untrusted Application will trigger TA installation via communication with a TAM. The TEEP Agent will interact with the TAM via a TEEP Broker that faciliates communications between the TAM and the TEEP Agent.

Some Trusted Component installation implementations might ask for a user's consent. In other implementations, a Device Administrator might choose what Untrusted Applications and related Trusted Components to be installed. A user consent flow is out of scope of the TEEP architecture.

The main components of the TEEP protocol consist of a set of standard messages created by a TAM to deliver Trusted Component 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. Consequently, Trusted Applications generally rely on a broker in the REE to provide access to network functionality in the REE. A broker does not need to know the actual content of messages to facilitate such access.

Similarly, since the TEEP Agent runs inside a TEE, the TEEP Agent generally relies on a TEEP Broker in the REE to provide network access, and relay TAM requests to the TEEP Agent and relay the responses back to the TAM.

5. Keys and Certificate Types

This architecture leverages the following credentials, which allow achieving end-to-end security between a TAM and a TEEP Agent.

Figure 4 summarizes the relationships between various keys and where they are stored. Each public/private key identifies a Trusted Component Signer, TAM, or TEE, and gets a certificate that chains up to some trust anchor. A list of trusted certificates is 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. In addition to the keys shown in Figure 4, there may be additional keys used for attestation or encryption. Refer to the RATS Architecture [I-D.ietf-rats-architecture] for more discussion.

Purpose	Cardinality & Location of Private Key	Private Key Signs	Location of Trust Anchor Store
Authenticating TEEP Agent	1 per TEE	TEEP responses	TAM
Authenticating TAM	1 per TAM	TEEP requests	TEEP Agent
Code Signing	1 per Trusted Component Signer	TA binary	TEE

Figure 4: Signature Keys

Note that Personalization Data is not included in the table above. The use of Personalization Data is dependent on how TAs are used and what their security requirements are.

TEEP requests from a TAM to a TEEP Agent are signed with the TAM private key (for authentication and integrity protection). Personalization Data and TA binaries can be encrypted with a key that is established with a content-encryption key established with the TEE public key (to provide confidentiality). Conversely, TEEP responses from a TEEP Agent to a TAM can be signed with the TEE private key.

The TEE key pair and certificate are thus used for authenticating the TEE to a remote TAM, and for sending private data to the TEE. 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 the TAM's Trust Anchor Store with the manufacturer certificates or CAs that are used to sign TEE keys. This is discussed further in Section 5.3 below. Typically the same key TEE pair is used for both signing and encryption, though separate key pairs might also be used in the future, as the joint security of encryption and signature with a single key remains to some extent an open question in academic cryptography.

The TAM key pair and certificate are used for authenticating a TAM to a remote TEE, and for sending private data to the TAM (separate key pairs for authentication vs. encryption could also be used in the future). A TAM provider is responsible for acquiring a certificate from a CA that is trusted by the TEEs it manages. This is discussed further in Section 5.1 below.

The Trusted Component Signer key pair and certificate are used to sign Trusted Components that the TEE will consider authorized to execute. TEEs must be configured with the certificates or keys that it considers authorized to sign TAs that it will execute. This is discussed further in Section 5.2 below.

5.1. Trust Anchors in a TEEP Agent

A TEEP Agent's Trust Anchor Store contains a list of Trust Anchors, which are typically 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 similarly to the Personalization Data for any other TA.

When Trust Anchor update is carried out, it is imperative that any update must maintain integrity where only an authentic Trust Anchor list from a device manufacturer or a Device Administrator is accepted. Details are out of scope of the architecture and can be addressed in a protocol document.

Before a TAM can begin operation in the marketplace to support a device with a particular TEE, it must be able to get its raw public key, or its certificate, or a certificate it chains up to, listed in the Trust Anchor Store of the TEEP Agent.

5.2. Trust Anchors in a TEE

The Trust Anchor Store in a TEE contains a list of Trust Anchors (raw public keys or certificates) that are used to determine whether TA binaries are allowed to execute by checking if their signatures can be verified. 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 similarly to the Personalization Data for any other TA, as discussed in Section 5.1.

5.3. Trust Anchors in a TAM

The Trust Anchor Store in a TAM consists of a list of Trust Anchors, which are certificates that sign various device TEE certificates. A TAM will accept a device for Trusted Component management if the TEE in the device uses a TEE certificate that is chained to a certificate or raw public key that the TAM trusts, is contained in an allow list, is not found on a block list, and/or fulfills any other policy criteria.

5.4. Scalability

This architecture uses a PKI (including self-signed certificates). Trust Anchors exist on the devices to enable the TEEP Agent to authenticate TAMs and the TEE to authenticate Trusted Component Signers, and TAMs use Trust Anchors to authenticate TEEP Agents. When 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 or all Trusted Component Signers that exist.

5.5. Message Security

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

These messages are signed end-to-end between a TEEP Agent and a TAM. Confidentiality is provided by encrypting sensitive payloads (such as Personalization Data and attestation evidence), rather than encrypting the messages themselves. Using encrypted payloads is important to ensure 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 installation of Trusted Components that the application needs to use.

An Untrusted Application might communicate with a TEEP Broker at runtime to trigger Trusted Component installation itself, or an Untrusted Application might simply have a metadata file that describes the Trusted Components it depends on and the associated TAM(s) for each Trusted Component, and an REE Application Installer can inspect this application metadata file and invoke the TEEP Broker to trigger Trusted Component 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 Trusted Component installation.

The Broker doesn't need to parse TEEP message content received from a TAM that should be processed by a TEE (see the ProcessTeepMessage API in Section 6.2.1). When a device has more than one TEE, one TEEP

Broker per TEE could be present in the REE or a common TEEP Broker could be used by multiple TEEs where the transport protocol (e.g., [I-D.ietf-teep-otrp-over-http]) allows the TEEP Broker to distinguish which TEE is relevant for each message from a TAM.

The TEEP Broker interacts with a TEEP Agent inside a TEE, and relays the response messages generated from the TEEP Agent back to the TAM.

The Broker only needs to return a (transport) error message to the TAM if the TEE is not reachable for some reason. Other errors are represented as TEEP response messages returned from the TEE which will then be passed to the TAM.

6.2. TEEP Broker Implementation Consideration

As depicted in Figure 5, there are multiple ways in which a TEEP Broker can be implemented, with more or fewer layers being inside the TEE. For example, in model A, the model with the smallest TEE footprint, only the TEEP implementation is inside the TEE, whereas the TEEP/HTTP implementation is in the TEEP Broker outside the TEE.

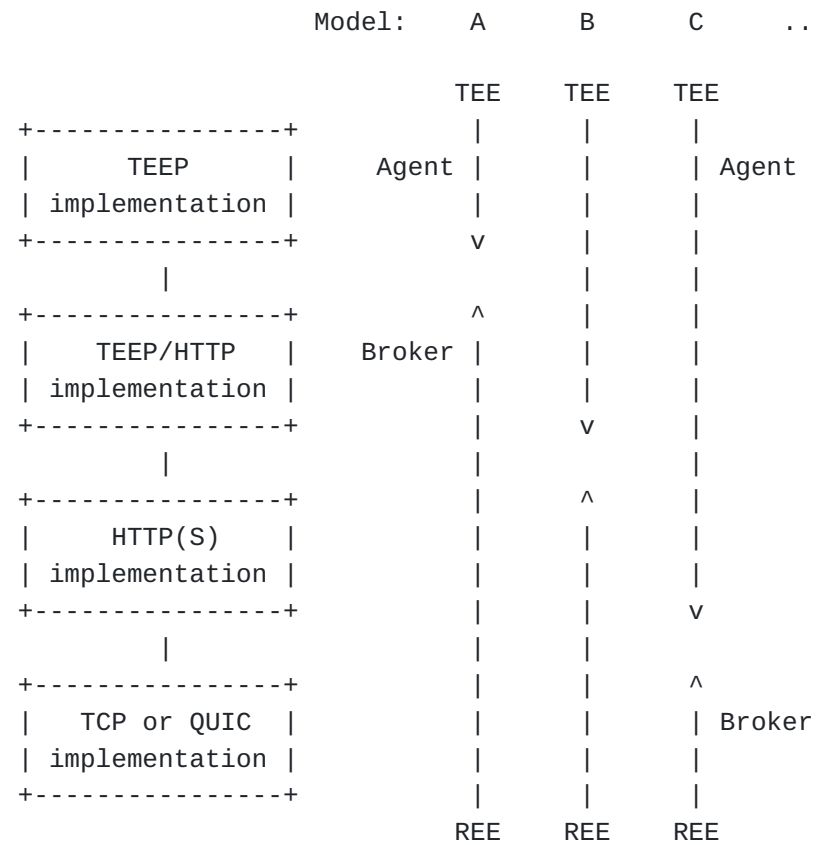

Figure 5: TEEP Broker Models

In other models, additional layers are moved into the TEE, increasing the TEE footprint, with the Broker either containing or

calling the topmost protocol layer outside of the TEE. An implementation is free to choose any of these models.

TEEP Broker implementers should consider methods of distribution, scope and concurrency on devices and runtime options.

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 an Untrusted Application) that it depends on a given Trusted Component, which may or may not already be installed in the TEE.

2. UnrequestTA: A notification from an REE application (e.g., an installer, or an Untrusted Application) that it no longer depends on a given Trusted Component, which may or may not already be installed in the TEE. For example, if the Untrusted Application is uninstalled, the uninstaller might invoke this conceptual API.

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

4. 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.

5. 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 a new TEEP session is being requested by a TEEP Agent.

2. ProcessTeepMessage: A message arriving on an existing TEEP session, 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.

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 require different degrees of confidence in 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. (See [I-D.ietf-rats-architecture] for more discussion.)

In TEEP, as depicted in Figure 6, the primary purpose of an attestation is to allow a device (the Attester) to prove to a TAM (the Relying Party) that a TEE in the device has particular properties, was built by a particular manufacturer, and/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.

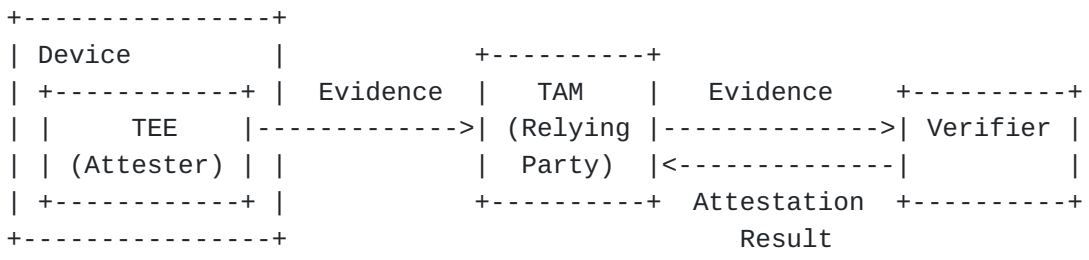

Figure 6: TEEP Attestation Roles

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 protocols. 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), as long as the format supports encryption of any information that is considered sensitive.

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. Note that the respective attestation algorithms are

not defined in the TEEP protocol itself; see [I-D.ietf-rats-architecture] and [I-D.ietf-teep-protocol] for more discussion.

There are a number of considerations that need to be considered when appraising evidence provided by a TEE, including:

- What security measures a manufacturer takes when provisioning keys into devices/TEEs;
- What hardware and software components have access to the attestation keys of the TEE;
- The source or local verification of claims within an attestation prior to a TEE signing a set of claims;
- The level of protection afforded to attestation keys against exfiltration, modification, and side channel attacks;
- The limitations of use applied to TEE attestation keys;
- The processes in place to discover or detect TEE breaches; and
- The revocation and recovery process of TEE attestation keys.

Some TAMs may require additional claims in order to properly authorize a device or TEE. 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.

For more discussion of the attestation and appraisal process, see the RATS Architecture [I-D.ietf-rats-architecture].

The following information is required for TEEP attestation:

- Device Identifying Information: Attestation information may need to uniquely identify a device to the TAM. Unique device identification allows the TAM to provide services 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. In some use cases it may be sufficient to identify only the class of the device. The security and privacy requirements regarding device identification will vary with the type of TA provisioned to the TEE.
- TEE Identifying Information: The type of TEE that generated this attestation must be identified. This includes version identification information for 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 address considerations around manufacturer provisioning, keying and support for the TEE.

*Freshness Proof: A claim that includes freshness information must be included, such as a nonce or timestamp.

8. Algorithm and Attestation Agility

RFC 7696 [RFC7696] outlines the requirements to migrate from one mandatory-to-implement cryptographic algorithm suite to another over time. This feature is also known as crypto agility. Protocol evolution is greatly simplified when crypto agility is considered during the design of the protocol. In the case of the 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. In the context of TEEP, symmetric algorithms are used for encryption and integrity protection of TA binaries and Personalization Data whereas the asymmetric algorithms are used for signing messages and managing symmetric keys.

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. Broker Trust Model

The architecture enables the TAM to communicate, via a TEEP Broker, with the device's TEE to manage Trusted Components. Since the TEEP Broker runs in a potentially vulnerable REE, the TEEP Broker could, however, be (or be infected by) malware. As such, all TAM messages are signed and sensitive data is encrypted such that the TEEP Broker cannot modify or capture sensitive data, but the TEEP Broker can still conduct DoS attacks as discussed in Section 9.3.

A TEEP Agent in a TEE is responsible for protecting against potential attacks from a compromised TEEP Broker or rogue malware in the REE. A rogue TEEP Broker might send corrupted data to the TEEP Agent, or launch a DoS attack by sending a flood of TEEP protocol requests, or simply drop or delay notifications to a TEE. The TEEP Agent validates the signature of each TEEP protocol request and

checks the signing certificate against its Trust Anchors. To mitigate DoS attacks, it might also add some protection scheme such as a threshold on repeated requests or number of TAs that can be installed.

Some implementations might rely on (due to lack of any available alternative) the use of an untrusted timer or other event to call the RequestPolicyCheck API (Section 6.2.1), which means that a compromised REE can cause a TEE to not receive policy changes and thus be out of date with respect to policy. The same can potentially be done by any other man-in-the-middle simply by blocking communication with a TAM. Ultimately such outdated compliance could be addressed by using attestation in secure communication, where the attestation evidence reveals what state the TEE is in, so that communication (other than remediation such as via TEEP) from an out-of-compliance TEE can be rejected.

Similarly, in most implementations the REE is involved in the mechanics of installing new TAs. However, the authority for what TAs are running in a given TEE is between the TEEP Agent and the TAM. While a TEEP Broker broker can in effect make suggestions, it cannot decide or enforce what runs where. The TEEP Broker can also control which TEE a given installation request is directed at, but a TEEP Agent will only accept TAs that are actually applicable to it and where installation instructions are received by a TAM that it trusts.

The authorization model for the UnrequestTA operation is, however, weaker in that it expresses the removal of a dependency from an application that was untrusted to begin with. This means that a compromised REE could remove a valid dependency from an Untrusted Application on a TA. Normal REE security mechanisms should be used to protect the REE and Untrusted Applications.

9.2. Data Protection

It is the responsibility of the TAM to protect data on its servers. Similarly, it is the responsibility of the TEE implementation to provide protection of data against integrity and confidentiality attacks from outside the TEE. TEEs that provide isolation among TAs within the TEE are likewise responsible for protecting TA data against the REE and other TAs. For example, this can be used to protect one user's or tenant's data from compromise by another user or tenant, even if the attacker has TAs.

The protocol between TEEP Agents and TAMs similarly is responsible for securely providing integrity and confidentiality protection against adversaries between them. It is a design choice at what layers to best provide protection against network adversaries. As

discussed in Section 6, the transport protocol and any security mechanism associated with it (e.g., the Transport Layer Security protocol) under the TEEP protocol may terminate outside a TEE. If it does, the TEEP protocol itself must provide integrity protection and confidentiality protection to secure data end-to-end. For example, confidentiality protection for payloads may be provided by utilizing encrypted TA binaries and encrypted attestation information. See [I-D.ietf-teep-protocol] for how a specific solution addresses the design question of how to provide integrity and confidentiality protection.

9.3. Compromised REE

It is possible that the REE of a device is compromised. We have already seen examples of attacks on the public Internet with billions of compromised devices being used to mount DDoS attacks. A compromised REE can be used for such an attack but it cannot tamper with the TEE's code or data in doing so. A compromised REE can, however, launch DoS attacks against the TEE.

The compromised REE may terminate the TEEP Broker such that TEEP transactions cannot reach the TEE, or might drop, replay, or delay messages between a TAM and a TEEP Agent. However, while a DoS attack cannot be prevented, the REE cannot access anything in the TEE if the TEE is implemented correctly. Some TEEs may have some watchdog scheme to observe REE state and mitigate DoS attacks against it but most TEEs don't have such a capability.

In some other scenarios, the compromised REE may ask a TEEP Broker to make repeated requests to a TEEP Agent in a TEE to install or uninstall a Trusted Component. An installation or uninstallation request constructed by the TEEP Broker or REE will be rejected by the TEEP Agent because the request won't have the correct signature from a TAM to pass the request signature validation.

This can become a DoS attack by exhausting resources in a TEE with repeated requests. In general, a DoS attack threat exists when the REE is compromised, and a DoS attack can happen to other resources. The TEEP architecture doesn't change this.

A compromised REE might also request initiating the full flow of installation of Trusted Components that are not necessary. It may also repeat a prior legitimate Trusted Component installation request. A TEEP Agent implementation is responsible for ensuring that it can recognize and decline such repeated requests. It is also responsible for protecting the resource usage allocated for Trusted Component management.

9.4. CA Compromise or Expiry of CA Certificate

A root CA for TAM certificates might get compromised or its certificate might expire, or a Trust Anchor other than a root CA certificate may also expire or be compromised. TEEs are responsible for validating the entire TAM certificate path, including the TAM certificate and any intermediate certificates up to the root certificate. See Section 6 of [RFC5280] for details. Such validation generally includes checking for certificate revocation, but certificate status check protocols may not scale down to constrained devices that use TEEP.

To address the above issues, a certificate path update mechanism is expected from TAM operators, so that the TAM can get a new certificate path that can be validated by a TEEP Agent. In addition, the Trust Anchor in the TEEP Agent's Trust Anchor Store may need to be updated. To address this, some TEE Trust Anchor update mechanism is expected from device OEMs, such as using the TEEP protocol to distribute new Trust Anchors.

Similarly, a root CA for TEE certificates might get compromised or its certificate might expire, or a Trust Anchor other than a root CA certificate may also expire or be compromised. TAMs are responsible for validating the entire TEE certificate path, including the TEE certificate and any intermediate certificates up to the root certificate. Such validation includes checking for certificate revocation.

If a TEE certificate path validation fails, the TEE might be rejected by a TAM, subject to the TAM's policy. To address this, some certificate path update mechanism is expected from device OEMs, so that the TEE can get a new certificate path that can be validated by a TAM. In addition, the Trust Anchor in the TAM's Trust Anchor Store may need to be updated.

9.5. Compromised TAM

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

9.6. Malicious TA Removal

It is possible that a rogue developer distributes a malicious Untrusted Application and intends to get a malicious TA installed. Such a TA might be able to escape from malware detection by the REE, or access trusted resources within the TEE (but could not access other TEEs, or access other TA's if the TEE provides isolation between TAs).

It is the responsibility of the TAM to not install malicious TAs in the first place. The TEEP architecture allows a TEEP Agent to decide which TAMs it trusts via Trust Anchors, and delegates the TA authenticity check to the TAMs it trusts.

It may happen that a TA was previously considered trustworthy but is later found to be buggy or compromised. In this case, the TAM can initiate the removal of the TA by notifying devices to remove the TA (and potentially the REE or Device Owner to remove any Untrusted Application that depend on the TA). If the TAM does not currently have a connection to the TEEP Agent on a device, such a notification would occur the next time connectivity does exist. That is, to recover, the TEEP Agent must be able to reach out to the TAM, for example whenever the RequestPolicyCheck API (Section 6.2.1) is invoked by a timer or other event.

Furthermore the policy in the Verifier in an attestation process can be updated so that any evidence that includes the malicious TA would result in an attestation failure. There is, however, a time window during which a malicious TA might be able to operate successfully, which is the validity time of the previous attestation result. For example, if the Verifier in Figure 6 is updated to treat a previously valid TA as no longer trustworthy, any attestation result it previously generated saying that the TA is valid will continue to be used until the attestation result expires. As such, the TAM's Verifier should take into account the acceptable time window when generating attestation results. See [I-D.ietf-rats-architecture] for further discussion.

9.7. TEE Certificate Expiry and 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 updates. On the other hand, it is imperative that OEMs or device providers plan for support of Trust Anchor update in their shipped devices.

For those cases where TEE devices are given certificates for which no good expiration date can be assigned the recommendations in Section 4.1.2.5 of [RFC5280] are applicable.

9.8. Keeping Secrets from the TAM

In some scenarios, it is desirable to protect the TA binary or Personalization Data from being disclosed to the TAM that distributes them. In such a scenario, the files can be encrypted

end-to-end between a Trusted Component Signer and a TEE. However, there must be some means of provisioning the decryption key into the TEE and/or some means of the Trusted Component Signer securely learning a public key of the TEE that it can use to encrypt. The Trusted Component Signer cannot necessarily even trust the TAM to report the correct public key of a TEE for use with encryption, since the TAM might instead provide the public key of a TEE that it controls.

One way to solve this is for the Trusted Component Signer to run its own TAM that is only used 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 Trusted Component manifest, determine there is a dependency with a TAM URI of the Trusted Component Signer's TAM. The Agent would then install the dependency, and then continue with the Trusted Component installation steps, including decrypting the TA binary with the relevant key.

9.9. REE Privacy

The TEEP architecture is applicable to cases where devices have a TEE that protects data and code from the REE administrator. In such cases, the TAM administrator, not the REE administrator, controls the TEE in the devices. As some examples:

* a cloud hoster may be the REE administrator where a customer administrator controls the TEE hosted in the cloud.

* a device manufacturer might control the TEE in a device purchased by a customer

The privacy risk is that data in the REE might be susceptible to disclosure to the TEE administrator. This risk is not introduced by the TEEP architecture, but is inherent in most uses of TEEs. This risk can be mitigated by making sure the REE administrator is aware of and explicitly chooses to have a TEE that is managed by another party. In the cloud hoster example, this choice is made by explicitly offering a service to customers to provide TEEs for them to administer. In the device manufacturer example, this choice is made by the customer choosing to buy a device made by a given manufacturer.

10. IANA Considerations

This document does not require actions by IANA.

11. Contributors

* Andrew Atyeo, Intercede (andrew.atyeo@intercede.com)

*Liu Dapeng, Alibaba Group (maxpassion@gmail.com)

12. Acknowledgements

We would like to thank Nick Cook, Minho Yoo, Brian Witten, Tyler Kim, Alin Mutu, Juergen Schoenwaelder, Nicolae Paladi, Sorin Faibish, Ned Smith, Russ Housley, Jeremy O'Donoghue, and Anders Rundgren for their feedback.

13. Informative References

[CC-Overview] Confidential Computing Consortium, "Confidential Computing: Hardware-Based Trusted Execution for Applications and Data", January 2021, <https://confidentialcomputing.io/wp-content/uploads/sites/85/2021/03/confidentialcomputing_outreach_whitepaper-8-5x11-1.pdf>.

[CC-Technical-Analysis] Confidential Computing Consortium, "A Technical Analysis of Confidential Computing, v1.2", October 2021, <https://confidentialcomputing.io/wp-content/uploads/sites/85/2022/01/CCC-A-Technical-Analysis-of-Confidential-Computing-v1.2.pdf>.

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

[GSMA] GSM Association, "GP.22 RSP Technical Specification, Version 2.2.2", June 2020, <https://www.gsma.com/esim/wp-content/uploads/2020/06/SGP.22-v2.2.2.pdf>.

[I-D.ietf-rats-architecture] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-18, 14 June 2022, <https://www.ietf.org/archive/id/draft-ietf-rats-architecture-18.txt>.

[I-D.ietf-suit-manifest] Moran, B., Tschofenig, H., Birkholz, H., and K. Zandberg, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", Work in Progress, Internet-Draft, draft-ietf-suit-manifest-17, 28

April 2022, <https://www.ietf.org/archive/id/draft-ietf-suit-manifest-17.txt>.

[I-D.ietf-teep-otrp-over-http]
Thaler, D., "HTTP Transport for Trusted Execution Environment Provisioning: Agent Initiated Communication", Work in Progress, Internet-Draft, draft-ietf-teep-otrp-over-http-13, 28 February 2022, <https://www.ietf.org/archive/id/draft-ietf-teep-otrp-over-http-13.txt>.

[I-D.ietf-teep-protocol] Tschofenig, H., Pei, M., Wheeler, D., Thaler, D., and A. Tsukamoto, "Trusted Execution Environment Provisioning (TEEP) Protocol", Work in Progress, Internet-Draft, draft-ietf-teep-protocol-08, 7 March 2022, <https://www.ietf.org/archive/id/draft-ietf-teep-protocol-08.txt>.

[OTRP] GlobalPlatform, "Open Trust Protocol (OTrP) Profile v1.1", GlobalPlatform GPD_SPE_123, July 2020, <https://globalplatform.org/specs-library/tee-management-framework-open-trust-protocol/>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

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

[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>.

[RFC9019] Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", RFC 9019, DOI 10.17487/RFC9019, April 2021, <https://www.rfc-editor.org/info/rfc9019>.

[SGX] Intel, "Intel(R) Software Guard Extensions (Intel (R) SGX)", n.d., <https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html>.

[TrustZone]
Arm, "Arm TrustZone Technology", n.d., <https://developer.arm.com/ip-products/security-ip/trustzone>.

Authors' Addresses

Mingliang Pei
Broadcom

Email: mingliang.pei@broadcom.com

Hannes Tschofenig
Arm Limited

Email: hannes.tschofenig@arm.com

Dave Thaler
Microsoft

Email: dthaler@microsoft.com

David Wheeler
Amazon

Email: davewhee@amazon.com