TEEP
Internet-Draft
Intended status: Informational
Expires: August 11, 2020

M. Pei
Symantec
H. Tschofenig
Arm Limited
D. Thaler
Microsoft
D. Wheeler
Intel

February 08, 2020

Trusted Execution Environment Provisioning (TEEP) Architecture draft-ietf-teep-architecture-06

Abstract

A Trusted Execution Environment (TEE) is an environment that enforces that only authorized code can execute within 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 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 http://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 August 11, 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction 3
2. Terminology 5
3. Use Cases 7
3.1. Payment 7
3.2. Authentication 7
3.3. Internet of Things 7
3.4. Confidential Cloud Computing 8
4. Architecture 8
4.1. System Components 8
4.2. Multiple TEEs in a Device 10
4.3. Multiple TAMs and Relationship to TAs 12
4.4. Untrusted Apps, Trusted Apps, and Personalization Data . 13
4.4.1. Examples of Application Delivery Mechanisms in Existing TEEs 14
4.5. Entity Relations 16
5. Keys and Certificate Types 17
5.1. Trust Anchors in a TEEP Agent 18
5.2. Trust Anchors in a TEE 19
5.3. Trust Anchors in a TAM 19
5.4. Scalability 19
5.5. Message Security 20
6. TEEP Broker 20
6.1. Role of the TEEP Broker 20
6.2. TEEP Broker Implementation Consideration 21
6.2.1. TEEP Broker APIs 21
6.2.2. TEEP Broker Distribution 22
7. Attestation 22

7.1. Information Required in TEEP Claims 24
8. Algorithm and Attestation Agility 24
9. Security Considerations 25
9.1. Broker Trust Model 25
9.2. Data Protection at TAM and TEE 25
9.3. Compromised REE 25
9.4. Compromised CA 26
9.5. Compromised TAM 26
9.6. Malicious TA Removal 26
9.7. Certificate Renewal 27
9.8. Keeping Secrets from the TAM 27
10. IANA Considerations 27
11. Contributors 27
12. Acknowledgements 27
13. Informative References 28
Authors' Addresses 28

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 execute applications in a protected environment that enforces that only authorized code can execute within that environment, 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).

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.

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. 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 trusted third 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 TAM (e.g., operated by 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 TAM 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 TA developer wants to remove a confidential TA from a device's TEE if the TA developer is no longer offering such TAs or the TAs

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 TA developer wants to define the relationship between cooperating TAs under the TA developer's control, and specify whether the TAs can communicate, share data, and/or share key material.

Note: The TA developer requires the help of a TAM to provision the Trusted Applications to remote devices and the TEEP protocol exchanges messages between a TAM and a TEEP Agent via a TEEP Broker.

2. Terminology

The following terms are used:

- 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 when it is a mobile device. The list of Trust Anchors is normally modifiable by the device's owner or Device Administrator. However the device manufacturer or network carrier (in the mobile device case) may restrict some modifications, for example, by not allowing the manufacturer or carrier's Trust Anchor to be removed or disabled.

- 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., 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.

- 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 (or more precisely, less trusted than the TEE).

- 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 Application (TA) Developer: An entity that wishes to provide functionality on devices that requires the use of one or more Trusted Applications.

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

- 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 a Rich Execution Environment.

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 unlocking the phone 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.

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

The Internet of Things (IoT) has been posing threats to critical infrastructure because of weak security in devices. 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 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. Full descriptions of components not previously defined are provided below. Interactions of all components are further explained in the following paragraphs.

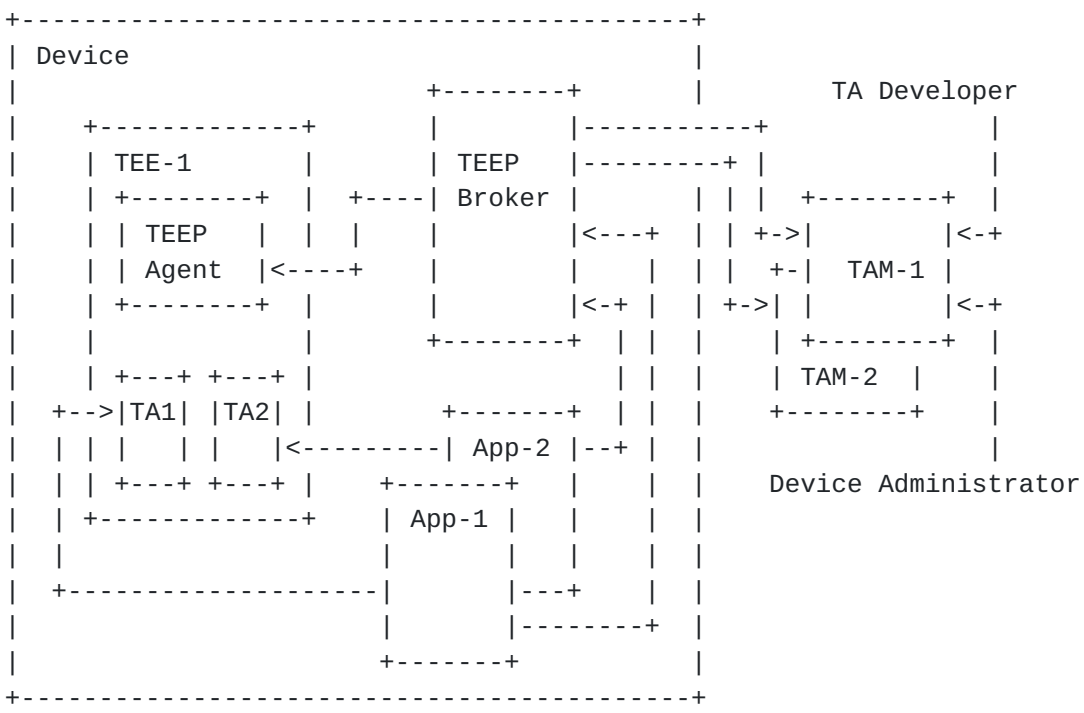


Figure 1: Notional Architecture of TEEP

- TA developers and Device Administrators utilize the services of a TAM to manage TAs on devices. TA developers 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 TAs on behalf of TA developers and Device Administrators. This includes creation and deletion of TAs, and may include, for example, over-the-air updates to keep TAs up-to-date and clean up when a version should be removed. TAMs may provide services that make it easier for TA developers 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 TAs 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. 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 TA developers, or a TAM may be private, and accessible by only one or a limited number of TA developers. It is expected that many manufacturers and network carriers will run their own private TAM.

  A TA developer 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 TA developer or Device Administrator may run their own TAM, but the devices they wish to manage must include this TAM's public key/ certificate, or a certificate it chains up to, in the Trust Anchor list.

  A TA developer or Device Administrator is free to utilize multiple TAMs. This may be required for a TA developer to manage multiple different types of devices from different manufacturers, or to manage mobile devices on different network 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 a device's Trust Anchor list. 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 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: 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, 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): Certificate-based credentials used for authenticating a device, a TAM and a TA developer. 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. 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) present themselves as separate containers within memory without a controlling manager within the TEE. As such, there might be multiple TEEP Brokers in the Rich Execution Environment, where each TEEP Broker communicates with one or more TEEs associated with it.

It is up to the Rich Execution Environment 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.

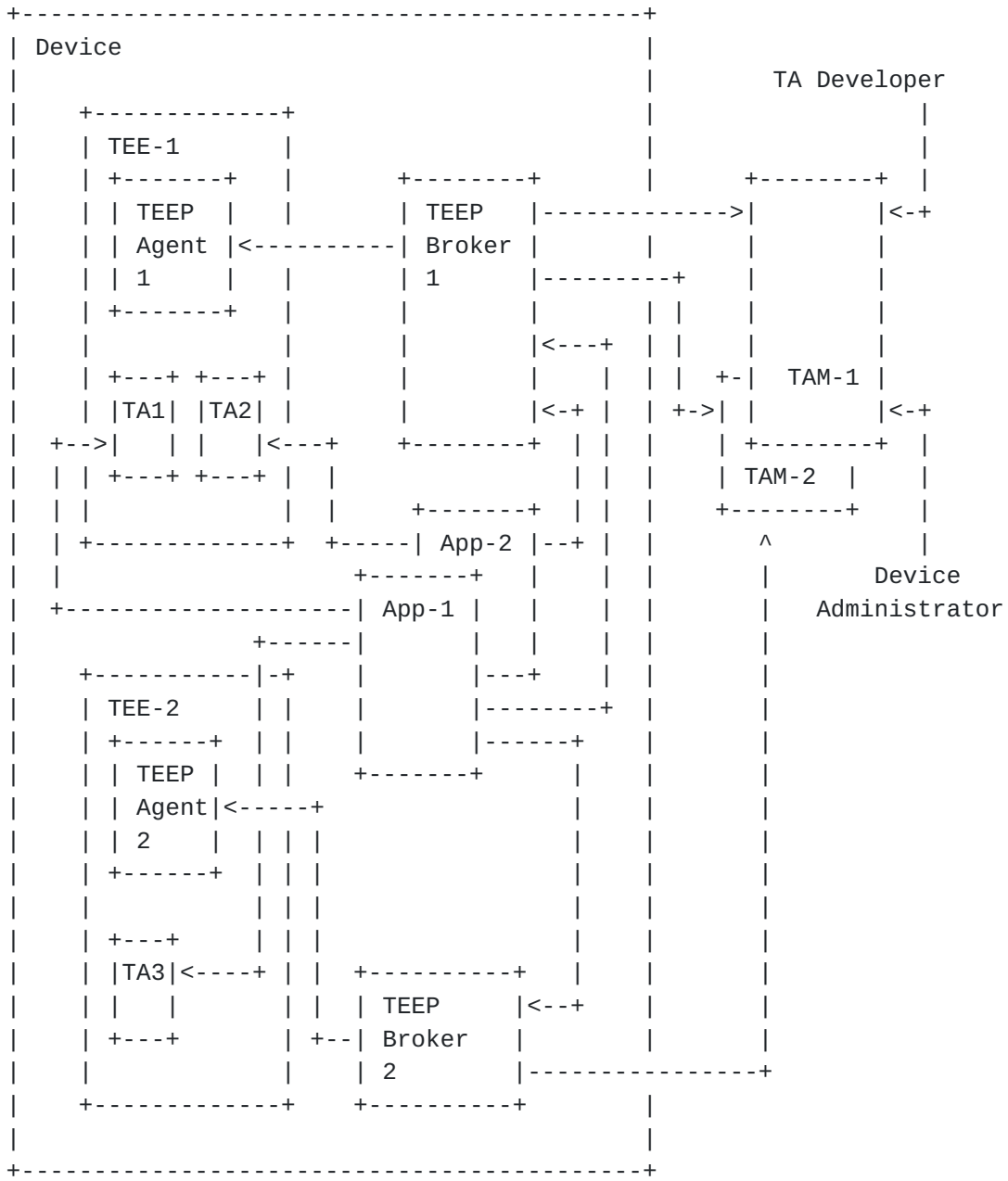


Figure 2: Notional Architecture of TEEP with multiple TEEs

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 TAs 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 communicate with might be made with or without input from an Untrusted Application, but is ultimately the decision of a TEEP Agent.

Each TA is digitally signed, protecting its integrity, and linking the TA back to the signer. The signer is usually the TA software author, but in some cases might be another party that the TA software author trusts, or a 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 TA author or signer selects one or more TAMs through which to offer their TA(s), and communicates the TA(s) to the TAM. In this document, we use the term "TA developer" to refer to the entity that selects a TAM and publishes a signed TA to it, independent of whether the publishing entity is the TA software author or the signer or both.

The TA developer chooses 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 TA developer 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 TA will be available through multiple TAMs.

When the developer of an Untrusted Application that depends on a TA 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 TA. In some situations, a TA may only be available via a single TAM - this is likely the case for enterprise applications or TA developers 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 can include multiple TAMs that support the required TA.

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 a TA, as well as any 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, TEEs compliant with GlobalPlatform 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 an Untrusted Application in a 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 require some additional data to personalize the TA to the TA developer or the device or a user. This personalization data is dependent on the TEE, the TA, and the TA developer; an example of personalization data might be a secret symmetric key used by the TA to communicate with the TA developer. The personalization data must be encrypted to preserve the confidentiality of potentially sensitive data contained within it. Other than this requirement to support confidentiality, the TEEP architecture places no limitations or requirements on the personalization data.

There are three possible cases for bundling of an Untrusted Application, TA(s), and personalization data:

1. The Untrusted Application, TA(s), and personalization data are all bundled together in a single package by a TA developer and provided to the TEEP Broker through the 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 TA developer'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 TA developer. Delivery of the TA and personalization data may be combined or separate.

The TEEP protocol treats each 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 binaries or personalization data.

4.4.1. 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 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.

In Arm TrustZone for A- and R-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, 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 any Untrusted Application, and so the complexity of Case 1 is lower than in the SGX example. Thus, Case 1 is possible as well, though still more complex than Cases 2 and 3.

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 of the TAM providers and TA binary signers; it may embed a list of default Trust Anchors into the TEEP Agent and TEE for TAM trust verification and TA signer verification.

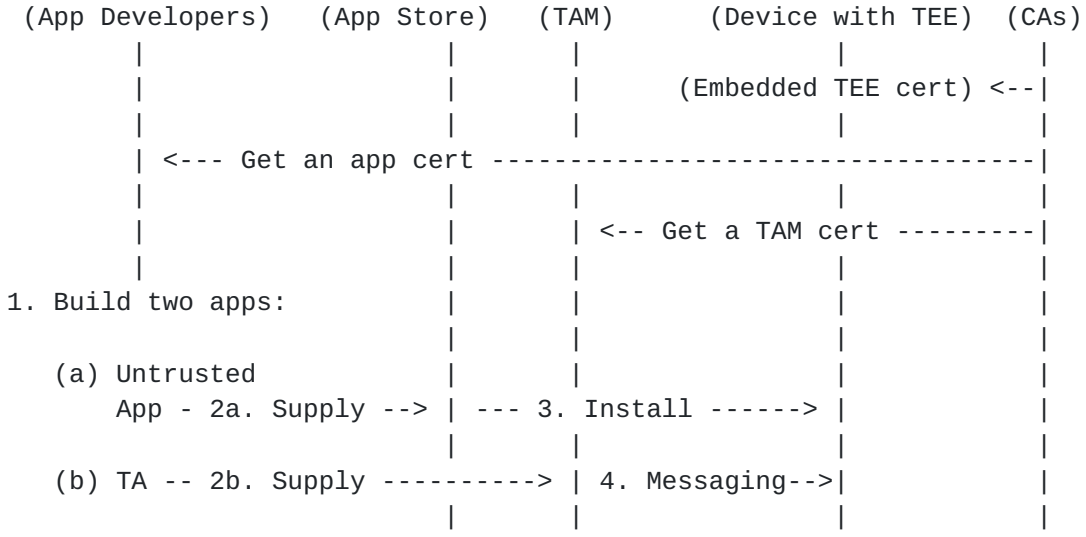


Figure 3: Developer Experience

Note that Figure 3 shows the TA developer as a TA signer. The TA signer is either the same as the TA developer, or is a related entity trusted to sign the developer's TAs.

Figure 3 shows an example where the same developer builds two applications: 1) an Untrusted Application; 2) a TA that provides some security functions to be run inside a TEE. At step 2, the TA developer uploads the Untrusted Application (2a) to an Application Store. The Untrusted Application may optionally bundle the TA binary. Meanwhile, the TA developer may provide its 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. Since the Untrusted Application depends on the TA, installing the Untrusted Application will trigger TA installation by initiating communication with a TAM. This is step 4. The TEEP Agent will interact with TAM via a TEEP Broker that faciliates communications between a TAM and the TEEP Agent in TEE.

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

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. Consequently, Trusted Applications generally rely on broker in the REE to provide access to nnetwork 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 delivering 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 TA developer, 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. In addition to the keys shown in Figure 4, there may be additional keys used for attestation. 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 TEE	1 per TEE	TEEP responses	TAM
Authenticating TAM	1 per TAM	TEEP requests	TEEP Agent
Code Signing	1 per TA developer	TA binary	TEE

Figure 4: 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.

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 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.

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. This is discussed further in Section 5.1 below.

The TA developer key pair and certificate are used to sign TAs 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 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 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 obtain a TAM certificate from a CA that is listed in the Trust Anchor Store of the TEEP Agent.

5.2. Trust Anchors in a TEE

A TEE determines whether TA binaries are allowed to execute by verifying whether the TA's signer chains up to a certificate in the TEE's Trust Anchor Store. 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, 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 TA management if the TEE in the device uses a TEE certificate that is chained to a certificate that the TAM trusts.

5.4. Scalability

This architecture uses a PKI, although self-signed certificates are also permitted. Trust Anchors exist on the devices to enable the TEE to authenticate TAMs and TA signers, and TAMs use Trust Anchors to authenticate TEEs. 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 TA developers that exist.

5.5. 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 a 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 the 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 only needs to return a (transport) 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

TEEP Broker implementers should consider methods of distribution, scope and concurrency on devices and runtime options. Several non-exhaustive options are discussed below.

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 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.

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. (See [I-D.ietf-rats-architecture] for more discussion.)

In TEEP, as depicted in Figure 5, 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.

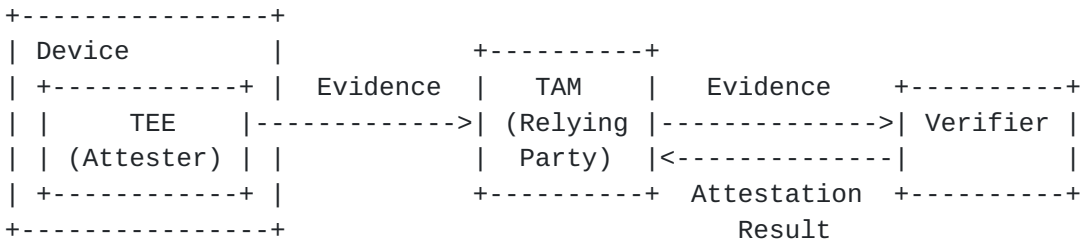


Figure 5: TEEP Attestation Roles

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 that 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 may need to uniquely identify a device to the TAM and TA developer. 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 info: The type of TEE that generated this attestation must be identified, including 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.

- Freshness Proof: A claim that includes freshness 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 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. 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. Broker Trust Model

The architecture enables the TAM to communicate, via a TEEP Broker, with the device's TEE to manage TAs. 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.

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. 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.

9.2. 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.

9.3. Compromised REE

It is possible that the REE of a device is compromised. If the REE is compromised, several DoS attacks may be launched. The compromised REE may terminate the TEEP Broker such that TEEP transactions cannot reach the TEE. However, while a DoS attack cannot be prevented, the REE cannot access anything in the TEE if it 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 have such 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 TA. A TA 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 TAs that are not necessary. It may also repeat a prior legitimate TA 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 TA management.

9.4. 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.

9.5. 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. It's the responsibility of the TAM to not install malicious trusted apps 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.

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.

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 a TA developer and a TEE. However, there must be some means of provisioning the decryption key into the TEE and/or some means of the TA developer securely learning a public key of the TEE that it can use to encrypt. One way to do this is for the TA developer to run its own TAM so that it can 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 TA developer'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. 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

[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/>.

[I-D.ietf-rats-architecture] Birkholz, H., Thaler, D., Richardson, M., and N. Smith, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-01 (work in progress), February 2020.

[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-03 (work in progress), February 2020.

[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>.

Authors' Addresses

Mingliang Pei
Symantec

EMail: mingliang_pei@symantec.com

Hannes Tschofenig
Arm Limited

EMail: hannes.tschofenig@arm.com

Dave Thaler
Microsoft

EMail: dthaler@microsoft.com

David Wheeler
Intel

EMail: david.m.wheeler@intel.com