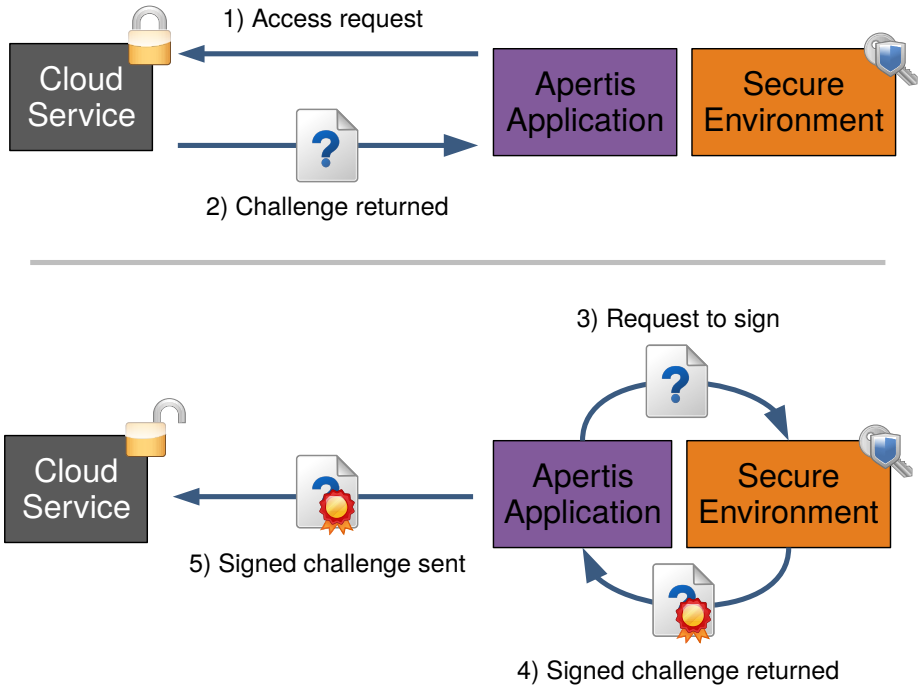
Integration of OP-TEE in Apertis

Contents

Some projects that wish to use Apertis have a requirement for strong security measures to be available in order to implement key system level functionality. A typical use case is enabling the decryption of protected content in such a way that doesn't allow the owner of the device doing the decryption to access the decryption keys. Another use for strong security is the protection of authentication keys. By shielding such keys within these strong security measures, it becomes much harder for the keys to be stolen and be used to impersonate the legitimate user.

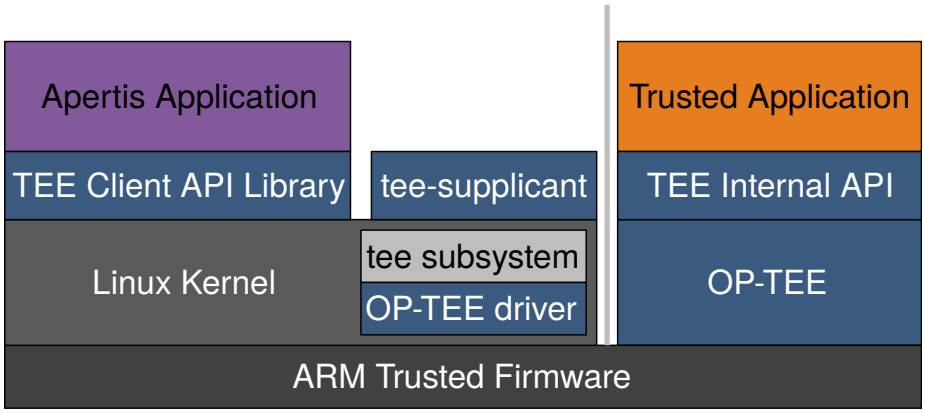


In the above example, when requesting access to the cloud service, the service returns a challenge response, which needs to be signed using asymmetric cryptography[1]. The Apertis application requests that functionality in the secure environment sign the challenge using a private key that it stores securely. The signed challenge is then returned to the cloud service, which checks the validity of the signature using the public key that it holds to authenticate the user. Such systems may additionally require the state of the system to be verified (typically by building a chain of trust[2]) before use of the secure keys is allowed, thus ensuring the device hasn't been altered in ways which may compromise protection of the keys.

Whilst a system could be architected to utilise a separate processor to perform such tasks, this significantly drives up system complexity and cost. Some platforms provide a mechanism to enable a secure, trusted environment or "Trusted Execution Environment[3]" (TEE) to be setup. A TEE runs on the application processor, but with mechanisms in place to isolate the code or data of the two running systems (the TEE and the main OS) from each other. ARM provides an implementation of such security mechanisms, known as ARM TrustZone[4], mainly on Cortex-A processors.

[1] https://en.wikipedia.org/wiki/Public-key_cryptography
[2] https://en.wikipedia.org/wiki/Chain_of_trust
[3] https://en.wikipedia.org/wiki/Trusted_execution_environment
[4] https://developer.arm.com/ip-products/security-ip/trustzone

System Architecture



A TEE exists as a separate environment running in parallel with the main operating system. At boot, both of these environments need to be loaded and initialised, this is achieved by running special boot firmware which enables the TrustZone security features and loads the required software elements. When enabled, a "secure monitor" runs in the highest privilege level provided by the processor. The secure monitor supports switching between the trusted and untrusted environments and enabling messages to be passed from one environment to the other. ARM provide a reference secure monitor as part of the ARM Trusted Firmware[5] (ATF) project. The ATF secure monitor provides an API to enable the development of trusted operating systems to run within the trusted environment, one such trusted OS is the open source OP-TEE[6]. OP-TEE provides a trusted environment which can run Trusted Applications (TAs), which are written against the TEE internal API.

As well as starting up a trusted OS in the trusted environment, ATF typically starts a standard OS such as Linux on the untrusted side, known as the rich operating system or "Rich Execution Environment" (REE), by running the firmware normally used for this OS. It is necessary for the OS to have drivers capable of interfacing with the secure monitor and that understands how to format messages for the trusted OS used on the trusted side. Linux contains a TEE subsystem[7] which provides a standardised way to communicate with TEE environments. The OP-TEE project have upstreamed a driver[8] to this subsystem to enable communications with the OP-TEE trusted environment.

OP-TEE relies on the REE to provide a number of remote services, such as file system access, as it does not have drivers for this functionality itself. The OP-

[5] https://github.com/ARM-software/arm-trusted-firmware
[6] https://www.op-tee.org/
[7] https://www.kernel.org/doc/Documentation/tee.txt
[8] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/tee/optee

TEE project provides a Linux user space supplicant daemon[9] which supplies the services required by the trusted environment. A library is also provided which implements a standardised mechanism, documented in the GlobalPlatform TEE Client API Specification v1.0[10], for communicating with the TEE. It is expected for this library to be used by applications needing to communicate with the TAs.

Boot Process

From a high level, the basic change required to the boot process is that the TEE need to be setup before the REE. The factor missing from this description is security. In order for the TEE to be able to achieve it's stated goal, providing a secure environment, it is necessary for the boot process to be able to guarantee that at least the setup of the TEE has not been tampered with. Such guarantees are provided by enabling secure boot for the relevant platform.

The process used to perform a secure boot is dependent on the mechanisms provided by the platform which vary from vendor to vendor. Typically it requires the boot process to be locked down to boot from known storage (such as a specific flash device) and for the boot binaries to be signed so that they can be verified at boot. The keys used for verification are usually read-only and held in fuses within the SoC.

The signed binaries comprise a series of bootloaders which progressively bring up the system, each being able to perform a bit more of the process utilising support enabled by earlier bootloaders. This series of bootloaders will load the secure monitor (known as `EL3 Runtime Software` in this context), OP-TEE (the `Secure-EL1 Payload`) and finally U-Boot (the `Non-trusted Firmware`), which loads Linux.

The ARMv8 architecture provides 4 privilege levels. The lowest privilege level, PL0, is used for executing user code under an OS or hypervisor. The next level, PL1, is used for running an OS like Linux, with PL2 above it available to run a hypervisor. The highest level PL3 is used for the secure monitor.

A more in-depth description of the boot process can be found in the OP-TEE documentation[11].

Trusted Applications

Trusted Applications (TAs) are applications that run within the trusted environment, on top of OP-TEE. Trusted Applications are used to provide the secured services and functionality that is needed in the platform. The TAs are identified by a UUID and are usually loaded from a file stored in the untrusted file system named after the UUID. In order to ensure the TAs haven't been tampered with they are signed. If the contents of the TA should remain protected, there are

[9] https://github.com/OP-TEE/optee_client
[10] https://globalplatform.org/specs-library/tee-client-api-specification/
[11] https://trustedfirmware-a.readthedocs.io/en/latest/design/firmware-design.html

options for storing it encrypted for further protection. Alternatively, if a TA is required before the tee-supplicant is running (and hence able to access the TA from the file system), TAs can also be built into the firmware as an early TA. A more in-depth description of TA implementation can be found in the OP-TEE documentation[12].

The OP-TEE project provides a number of TA examples[13].

Trusted Applications provide immense flexibility in the functionality that can be provided from the TEE environment. This flexibility is such that a proof of concept has been completed implementing a TPM 2.0 implementation[14] that can be used in OP-TEE[15].

Virtualisation Support

As the hypervisor and secure monitor each have a separate privilege level, it is possible for the TEE to co-exist with systems running a hypervisor. OP-TEE currently has experimental support[16] for the XEN hypervisor running on an emulated ARMv8 system. The current approach provides a separate context for each of the Virtual Machines (VMs) running on the hypervisor. This requires support from the hypervisor to enable communication between the Virtual Machines (VM) running on the hypervisor and the TEE and to ensure the TEE is using the context associated with the calling VM. The experimental support currently disables access to hardware resources, such as cryptographic engines, in the TEE as a mechanism to share such resources safely between the separate TEE contexts has not yet been created.

Enabling TEE in Apertis

Apertis does not provide the vast majority of the functionality needed to implement a TEE. A number of steps need to be taken in order to enable TEE support in Apertis.

Secure Boot

Secure boot provides an initial important step in initialisation of the TEE by ensuring that the initialisation process is able to proceed without interference. Unfortunately this fundamental step is very platform dependent and can not be solved as a general case. Apertis has already taken steps to document and demonstrate secure boot[17]. At the moment, Apertis only ships some support for secure on the SABRE Lite platform. This provides a good reference for the

[12] https://optee.readthedocs.io/en/latest/architecture/trusted_applications.html
[13] https://github.com/linaro-swg/optee_examples
[14] https://github.com/Microsoft/ms-tpm-20-ref
[15] https://github.com/jbech-linaro/manifest/tree/ftpm
[16] https://optee.readthedocs.io/en/latest/architecture/virtualization.html
[17] https://sjoerd.pages.apertis.org/apertis-website/architecture/secure-boot/

overall process but, unfortunately, the SABRE Lite is not a good choice as a technology demonstrator for TEE due to its age.

We advise the implementation of a TEE demonstrator on a more modern platform to take advantage of the more advanced functionality found in such platforms. This will be covered in more detail in our recommendations for the next steps.

In addition to the board verifying the initial binaries that are executed, it is important that the verification of binaries continues through the boot process in order to build a chain of trust[18] so that later stages can determine whether boot was carried out appropriately.

ARM Trusted Firmware

The current ARM Trusted Firmware package in Debian does not build for any platforms currently supported in Apertis. The package will need to be tweaked to sign the ATF binaries using an Apertis key. In order to support ATF in Apertis, one of the following options will need to be taken:

- Adopt a platform already supported by the build as an additional platform in Apertis
- Enable support for a platform supported by ATF but not currently built by the deb packaging
- Add support for a preferred platform to ATF and enable it in the packaging

From the perspective of enabling ATF, these are broadly in order of effort, though clearly adding an additional platform to Apertis increases the effort for ongoing baseline maintenance.

Requirements

In order to implement Trusted Board Boot[19] it will be necessary to upgrade `mbedtls`. This functionality is likely to be considered critical by project developers.

OP-TEE OS

The OP-TEE project provides the OP-TEE OS[20] as the trusted OS that runs in the TEE. This is not currently packaged for Debian and it would need to be to incorporated into Apertis. Like ATF, an Apertis key will need to be used to sign the binaries intended for the TEE to ensure the chain of trust. Currently when OP-TEE is built, it embeds the public key that will be used for verifying TAs. As with the key/keys used in other steps of this process, in order to ensure

[18] https://en.wikipedia.org/wiki/Chain_of_trust
[19] https://trustedfirmware-a.readthedocs.io/en/latest/design/trusted-board-boot.html
[20] https://github.com/OP-TEE/optee_os

that products are properly secured, would be necessary for product teams to at a minimum replace the key used with a product specific one. A product team may wish to modify OP-TEE to support alternative key management solutions, this is expected by the OP-TEE developers[21].

In addition to the trusted OS, the build of the OP-TEE OS source also builds the TA-devkit. The TA-devkit provides the resources necessary to both build and sign TAs. The TA-devkit will need to be packaged so that it can be provided as a build dependency for any TAs.

Linux Kernel

Debian (and thus the Apertis config) does already enable the TEE subsystem on arm64 where ATF can be used. It is understood that this should be sufficient and thus no extra modifications to the kernel will be required.

OP-TEE Supplicant and User Space Libraries

In addition to the trusted OS, the OP-TEE project provides the OP-TEE supplicant and TEE Client API[22]. The supplicant provides services to OP-TEE that it does not directly provide itself and the TEE Client API provides a user space API in the REE to communicate with the TEE. As with the OP-TEE OS, these components are currently not packaged for Debian and would need to be. As these components run in the REE they don't need to be signed.

Sample TAs

To enable early investigation of TEEs on Apertis, the example TAs[23] should be packaged. For simple use cases, it may be that these examples either fulfil or provide a framework for development of the TEE requirements. They will provide a useful reference of how to package TAs for Apertis even for the use cases that are not covered by the examples.

The sample TAs will be signed with the key provided by the Apertis TA-devkit package (which will be a build dependency) and thus will be usable with the OP-TEE OS built for Apertis.

Test Suite

A test suite[24] exists for OP-TEE. Providing this in Apertis would enable developers to gain some confidence that OP-TEE was installed and initialised correctly.

[21] https://github.com/OP-TEE/optee_os/issues/2233#issuecomment-379253182
[22] https://github.com/OP-TEE/optee_client
[23] https://github.com/linaro-swg/optee_examples
[24] https://github.com/OP-TEE/optee_test

Debos Scripting

Once components are added to the Apertis project, we need a way to combine them into an image that can be booted on the target platform. In Apertis this is performed by Debos using configuration files to determine exactly what packages are added to each image. This also allows for the images to be built automatically and regularly using the latest versions of packages. A special image to automate configuration of the boot process can also be generated like the one provided to update the U-Boot bootloader for the i.MX6 SABRE Lite board[25].

Next Steps

Integrating OP-TEE into Apertis substantially alters the boot process and requires secure boot to be working effectively to be valuable.

Whilst the research carried out to write this proposal has attempted to consider the impacts of adding this support to Apertis, there remains a risk that some potential issues have gone unnoticed. We therefore advise following up this document with adding the support to Apertis for at least one reference platform so that the basic components are formally integrated into Apertis; to provide as a solid reference for product teams and further lower the risk of Apertis adoption for teams wishing to use OP-TEE.

Choice of Reference Platform

The OP-TEE project is specifically targeted towards the ARM ecosystem, specifically those that provide ARM TrustZone. ARM TrustZone has been improved in later iterations of the technology and standardised with a reference implementation available to for using TEEs as part of the ATF project. We recommend that a platform that is capable of utilising ATF is chosen for this reference. An advantage of implementing the TEE using ATF is that this provides a standardised interface for the trusted OS and thus allows Apertis to potentially be used with alternative trusted OS implementations.

Test Integration

The availability of a test suite provides some coverage of the OP-TEE functionality with minimal effort as this should be usable from automated testing.

Whilst the test suite will test operation of OP-TEE itself, an important part of initialising a TEE is the platform specific secure boot. Unless using a platform very closely aligned with an Apertis reference platform, this step will be the responsibility of the product team. To ensure that this is properly implemented,

[25] https://gitlab.apertis.org/infrastructure/apertis-image-recipes/-/blob/apertis/v2021dev2/mx6qsabrelite-uboot-installer.yaml

tests could be developed that attempt to utilise incorrectly signed binaries at the different stages of the boot process to ensure that each step is properly validated, providing a reference for how to test secure boot.

Experience with the SABRE Lite has shown that whilst devices may be set up to emulate a secured configuration, their behavior differs from the behavior of devices locked via its embedded fuses. Since boards locked in a secure boot configuration no longer allow some operations, they become less useful for general development. For this reason, a dedicated set of boards locked via fuses may be required to fully test that secure boot restrictions are being enforced.