



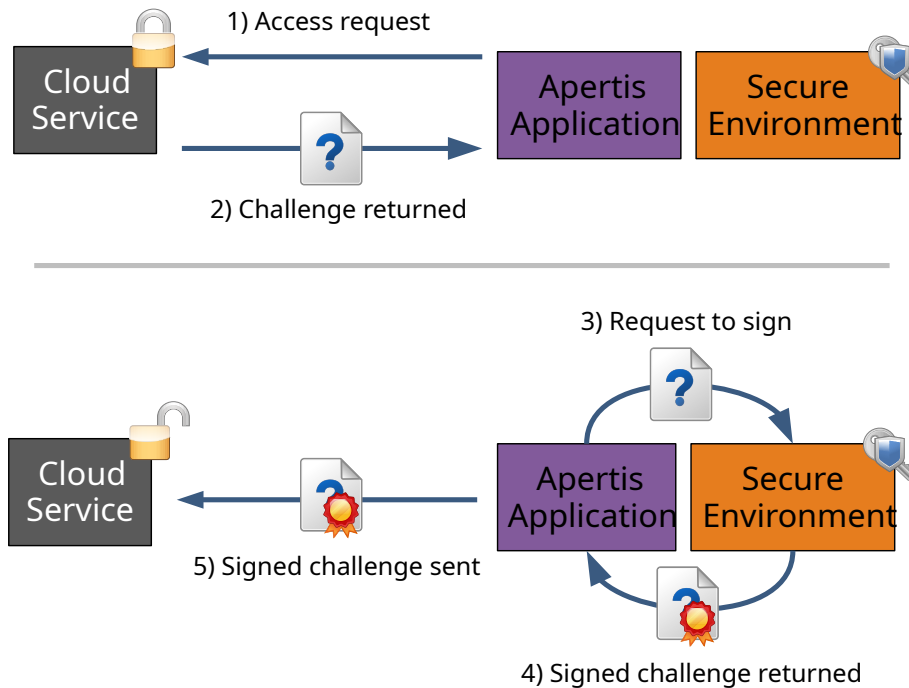
Integration of OP-TEE in Apertis

Contents

System Architecture	4
Boot Process	5
Trusted Applications	5
Virtualization Support	6
Container Support	7
Advanced topology support	8
Proxying TEE Access	8
Low impact hypervisor support for TEE	11
Enabling TEE in Apertis	12
Reference Platform Selection	13
Core components	14
Secure Boot	14
ARM Trusted Firmware	14
OP-TEE OS	15
Linux Kernel	15
OP-TEE Supplciant and User Space Libraries	15
Sample TAs	15
Debos Scripting	16
Hypervisor Integration	16
Xen Hypervisor	16
Linux Kernel	17
Dual configuration build of OP-TEE	17
TEE Proxy	17
Low Impact Hypervisor Support	17

Test Integration 18

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.



35

36 In the above example, when requesting access to the cloud service, the service
 37 returns a challenge response, which needs to be signed using [asymmetric cryptography](https://en.wikipedia.org/wiki/Public-key_cryptography)¹. The Apertis application requests that functionality in the secure
 38 environment sign the challenge using a private key that it stores securely. The
 39 signed challenge is then returned to the cloud service, which checks the valid-
 40 ity of the signature using the public key that it holds to authenticate the user.
 41 Such systems may additionally require the state of the system to be verified
 42 (typically by building a [chain of trust](https://en.wikipedia.org/wiki/Chain_of_trust)²) before use of the secure keys is allowed,
 43 thus ensuring the device hasn't been altered in ways which may compromise
 44 protection of the keys.
 45

46 Whilst a system could be architected to utilize a separate processor to perform
 47 such tasks, this significantly drives up system complexity and cost. Some plat-
 48 forms provide a mechanism to enable a secure, trusted environment or “[Trusted Execution Environment](https://en.wikipedia.org/wiki/Trusted_execution_environment)”³(TEE) to be setup. A TEE runs on the application
 49 processor, but with mechanisms in place to isolate the code or data of the two
 50 running systems (the TEE and the main OS) from each other. ARM provides
 51 an implementation of such security mechanisms, known as [ARM TrustZone](https://developer.arm.com/ip-products/security-ip/trustzone)⁴,
 52 mainly on Cortex-A processors.
 53

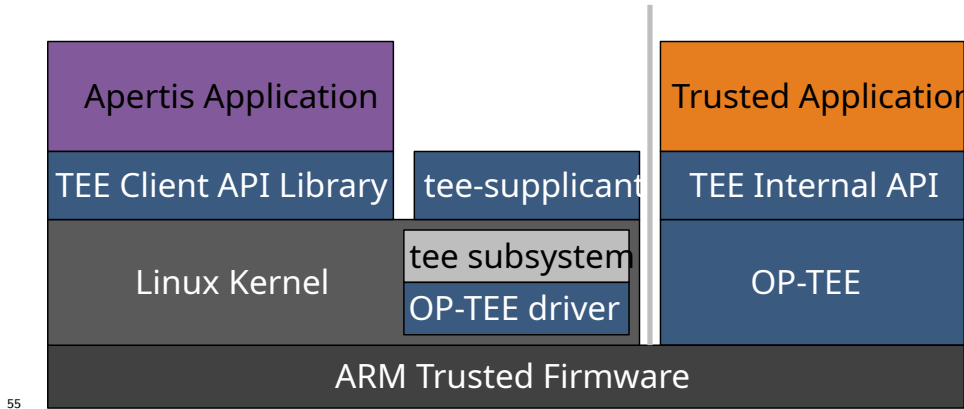
¹https://en.wikipedia.org/wiki/Public-key_cryptography

²https://en.wikipedia.org/wiki/Chain_of_trust

³https://en.wikipedia.org/wiki/Trusted_execution_environment

⁴<https://developer.arm.com/ip-products/security-ip/trustzone>

54 System Architecture



56 A TEE exists as a separate environment running in parallel with the main op-
 57 erating system. At boot, both of these environments need to be loaded and
 58 initialized, this is achieved by running special boot firmware which enables the
 59 TrustZone security features and loads the required software elements. When
 60 enabled, a “secure monitor” runs in the highest privilege level provided by the
 61 processor. The secure monitor supports switching between the trusted and
 62 untrusted environments and enabling messages to be passed from one environ-
 63 ment to the other. ARM provide a reference secure monitor as part of the [ARM
 64 Trusted Firmware](https://github.com/ARM-software/arm-trusted-firmware)⁵ (ATF) project. The ATF secure monitor provides an API to
 65 enable the development of trusted operating systems to run within the trusted
 66 environment, one such trusted OS is the open source [OP-TEE](https://www.op-tee.org/)⁶. OP-TEE pro-
 67 vides a trusted environment which can run **Trusted Applications** (TAs), which
 68 are written against the TEE internal API.

69 As well as starting up a trusted OS in the trusted environment, ATF typi-
 70 cally starts a standard OS such as Linux on the untrusted side, known as the
 71 rich operating system or “Rich Execution Environment”(REE), by running the
 72 firmware normally used for this OS. It is necessary for the OS to have drivers
 73 capable of interfacing with the secure monitor and that understands how to
 74 format messages for the trusted OS used on the trusted side. Linux contains
 75 a [TEE subsystem](https://www.kernel.org/doc/Documentation/tee.txt)⁷ which provides a standardized way to communicate with
 76 TEE environments. The OP-TEE project have upstreamed a [driver](https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/tee/op-tee)⁸ to this
 77 subsystem to enable communications with the OP-TEE trusted environment.

78 OP-TEE relies on the REE to provide a number of remote services, such as file
 79 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/op-tee>

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

85 Boot Process

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

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

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

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

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

110 Trusted Applications

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

⁹https://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>

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

122 The OP-TEE project provides a number of [TA examples](#)¹³.

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

127 Virtualization Support

128 As the hypervisor and secure monitor each have a separate privilege level, it is
129 possible for the TEE to co-exist with systems running a hypervisor. Whilst it
130 is possible for the two to exist, a number of adaptations need to be made to allow
131 communications to happen.

132 When running on a hypervisor, the guest OS uses intermediate physical ad-
133 dresses (IPAs) rather than physical memory addresses. These IPAs are then
134 translated by the hypervisor to real physical addresses. The TEE concept was
135 not developed with hypervisors in mind and the REE expects to pass the mem-
136 ory regions it uses for communicating with the TEE as physical addresses. How-
137 ever, unlike the TEE, the guest OS (acting as the REE) does not have access
138 to the actual physical addresses, which will lead to miss-communication as to
139 where data is stored. The hypervisor also needs to know that the contents of the
140 used regions of physical memory can't be swapped out whilst communication
141 between the TEE and REE is on going. Additionally something would need to
142 keep track of which VM made the request so the response can be passed back
143 to the right VM and handle situations such as the VM dying whilst the TEE
144 was handling a request.

145 It is therefore necessary for both the TEE and hypervisor to be modified for
146 things to function. Virtualization support has already been added to OP-TEE
147 and [experimental support](#)¹⁶ has been added to the Xen hypervisor running on an
148 emulated ARMv8 system. The current approach modifies OP-TEE to provide
149 a common TEE infrastructure with separate TEE contexts made available for
150 each of the Virtual Machines (VMs) in which the trusted applications for each
151 VM run.

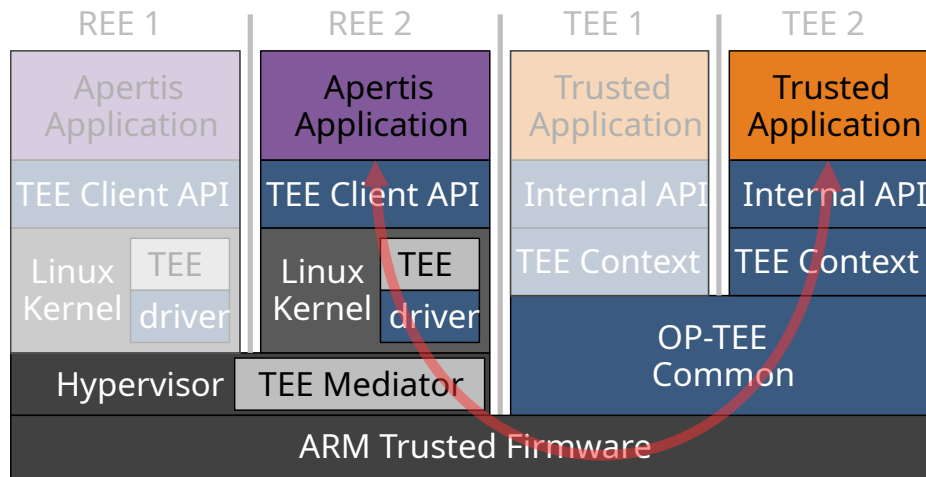
¹²https://optee.readthedocs.io/en/latest/architecture/trusted_applications.html

¹³https://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>



152

153 This works in conjunction with support from the hypervisor to provide memory
 154 mapping and to provide enumeration of the VMs so that the right context is
 155 used for each VM. The functionality added to the hypervisor is called the “TEE
 156 mediator”. The advantage of this approach is that, whilst it requires changes
 157 the TEE and hypervisor, it keeps the API as seen by the guest OS and trusted
 158 applications the same, and thus existing applications and TAs do not need to
 159 be made aware of the virtualization used on the platform.

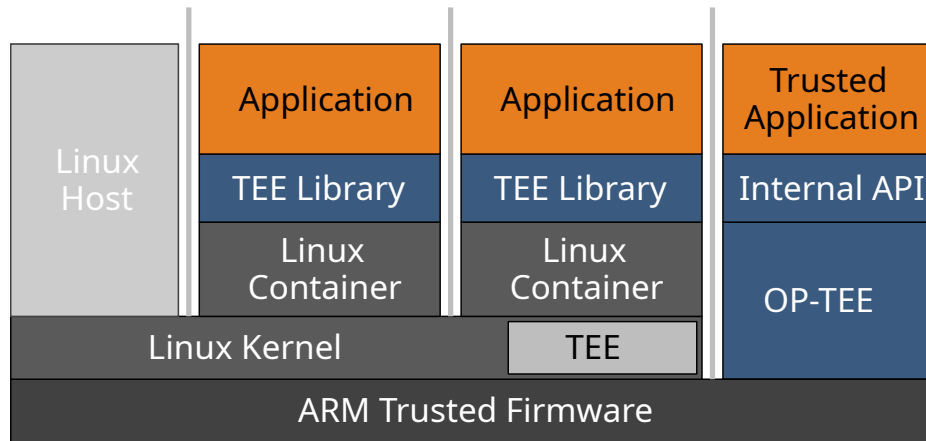
160 The virtualization support currently available in OP-TEE is configured at
 161 build time. A build supporting virtualization will currently not work in
 162 non-virtualized environments and vice versa. Thus separate builds will be
 163 required for each of these instances.

164 Access to hardware resources, such as cryptographic engines, has not been made
 165 virtualization aware and thus can’t currently be handled safely between TEE
 166 contexts nor ensure that data from these resources isn’t leaked between TEE
 167 contexts or to the REEs. Thus such hardware is **not currently supported when**
 168 **virtualization is enabled**¹⁷ in OP-TEE.

169 Container Support

170 Containers, such as LXC containers, provide a significant level of isolation be-
 171 tween the processes running in each container and the main host operating
 172 system, however each of these environments utilizes a shared kernel. Unless ex-
 173 plicitly denied (such as by using LXC’s cgroup support) each of the containers
 174 will have the ability to make calls to the TEE.

¹⁷<https://optee.readthedocs.io/en/latest/architecture/virtualization.html?highlight=virtualization#sharing-hardware-resources-and-ptas>



175

176 We expect that calls between the containers and TEE would work, however un-
 177 like when virtualization is utilized, the TEE will not provide a separate context
 178 for each of the REEs and as such it is likely that, depending on the trusted
 179 applications installed, it might be possible for some data to leak between the
 180 containers via the TEE.

181 More investigation work is required to understand the impact of this topology
 182 on the tee-supplciant. It is currently unclear whether it would be possible to
 183 have more than one instance of the tee-supplciant. Running the tee-supplciant
 184 on the host may be an option, though this may provide a way for containers
 185 to bypass their containment, by getting the supplciant to perform operations
 186 on or from the host. Additionally, the tee-supplciant can be **extended with**
 187 **plugins**¹⁸. For this to function, the tee-supplciant would realistically need to be
 188 in the container and thus a tee-supplciant would probably be needed by each
 189 container that needed one.

190 In the event that the TEE can not be directly utilized by containers it will be
 191 possible to utilize a TEE proxy as described later.

192 Advanced topology support

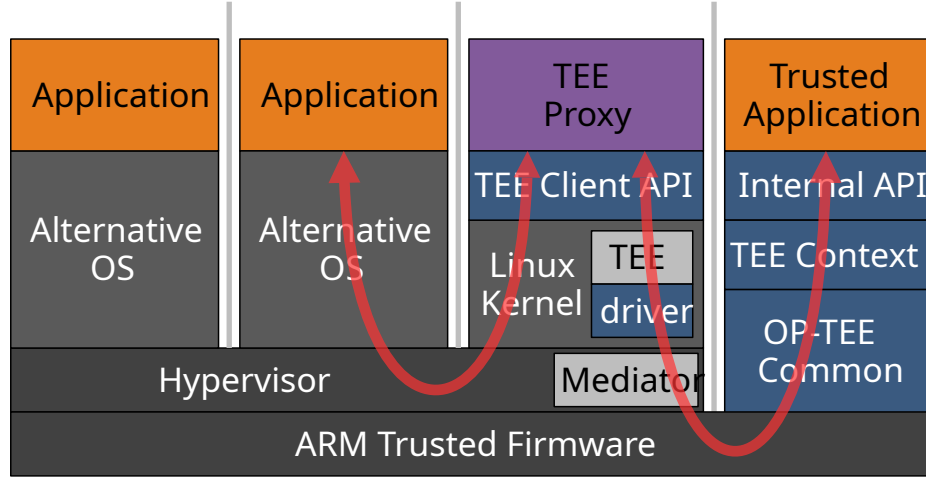
193 It is expected that product teams will want to utilize topologies that go beyond
 194 what is currently available in OP-TEE. Below we cover a few strategies that we
 195 envisage could be implemented.

196 Proxying TEE Access

197 It is expected that some systems will require non-Apertis guests to be run
 198 alongside Apertis on a hypervisor. These non-Apertis guests may also lack

¹⁸https://github.com/linaro-swg/optee_examples/commit/0607ed40746afe4cb8993149a6f275df648f7bad#diff-92cd10e6b8b068a931196d1d73a032543d5ca1a5bf445e27a1af74258254517cA

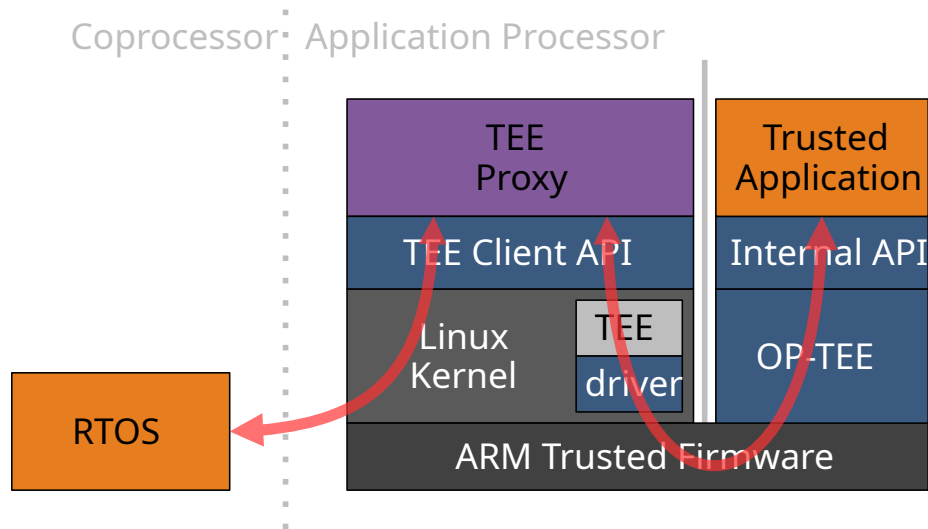
199 support for TEEs or there may be a desire for all the guest environments to be
 200 able to access a shared TEE environment. For these instances we would suggest
 201 the development of a “TEE proxy”. The TEE proxy would be a daemon, which
 202 provides proxied access to TEE functionality.



203
 204 The other virtualized environments would utilize a standard mechanism to com-
 205 municate with the TEE proxy. In order for such a topology to be viable care
 206 would need to be taken to ensure that communications with the TEE proxy
 207 could be authenticated and secured. We recommend that a TCP based network
 208 connection be used to communicate with the TEE proxy. This would allow stan-
 209 dard security measures that are widely implemented and understood (such as
 210 SSL) to be used for communications. This could be implemented over a virtual
 211 network link provided by the hypervisor. Additionally it would be necessary
 212 for the guest environment hosting the TEE proxy to be trusted as it will have
 213 access to the communications between the other VMs and the TEE.

214 It exact serialisation used over this network link could be D-Bus, though gRPC
 215 or other RPC mechanism may be a better fit.

216 Such a TEE proxy could also be used in other instances, such as when access
 217 to TEE functionality is required by processes running on a separate core, such
 218 as a coprocessor, which will not have any access to the TEE environment.

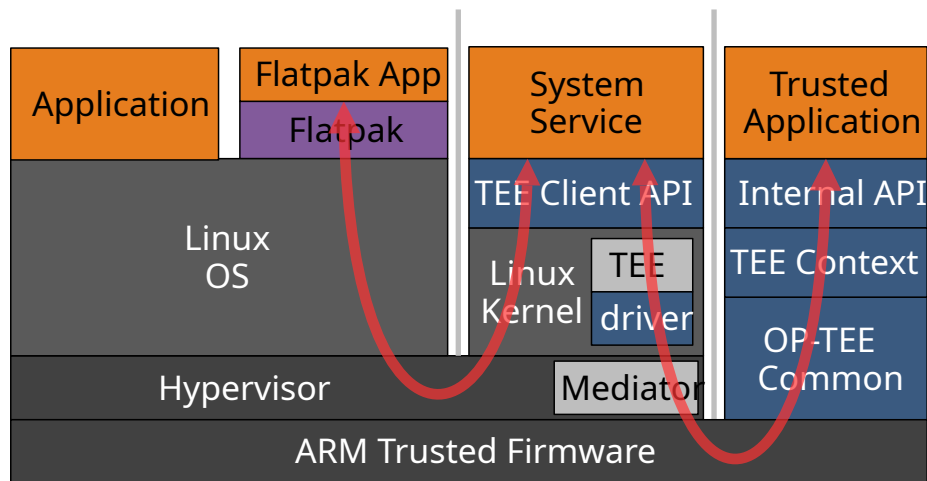


219

220 As before, such a topology would require the TEE proxy to authenticate re-
 221 quests from the coprocessor, the communications mechanism used would need
 222 to be secure and the TEE proxy trusted. As the interface between co-processors
 223 and the rest of the system varies wildly, it is hard to predict exactly how commu-
 224 nications would be structured. If the co-processor is connected via a serial link,
 225 then the [Point-to-Point Protocol](https://en.wikipedia.org/wiki/Point-to-Point_Protocol)¹⁹ (PPP) could be used to enable network con-
 226 nectivity. In other cases the TEE proxy may need to be customized to support
 227 communications with the co-processor.

228 A similar but slightly different use case would be where containerized applica-
 229 tions are used, such as Flatpaks. In these instances we suggest that a system
 230 service exposing higher level functionality than that expected by a TEE proxy
 231 would be more appropriate.

¹⁹https://en.wikipedia.org/wiki/Point-to-Point_Protocol

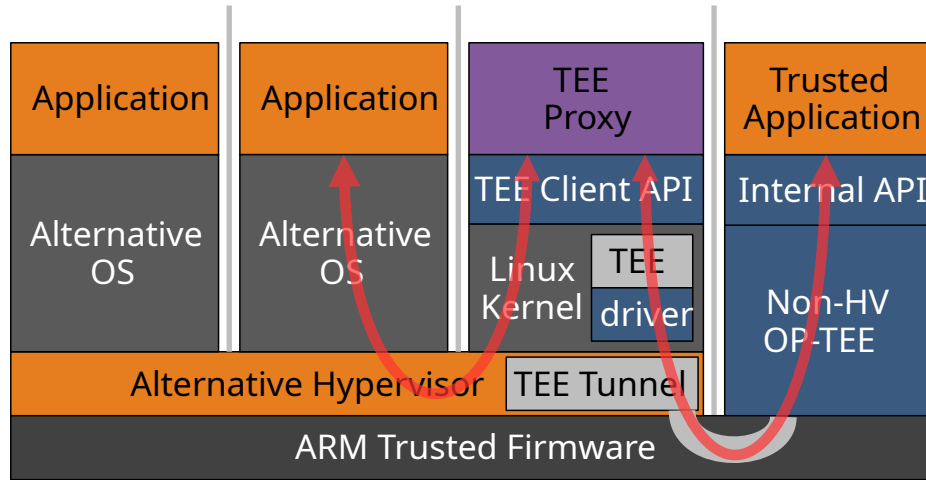


For example, rather than each Flatpak downloading encrypted downloads and passing the content of these to a TA to decrypt via a TEE proxy, a system service could be provided to download, validate and decrypt files for the Flatpak apps. This would be advantageous as it would reduce the number of hops that data would need to take from being downloaded to being available in the container decrypted, improving device efficiency.

As with the previous topologies it would still require the system service to be trusted, the communication method used between the containerized application and the system service, such as a virtual network link, would need to be secure and communications with the system service authenticated. We expect such a component to be quite user case specific. As a result we do not expect the Apertis project to provide an example system service, though expect that the TEE proxy would serve as a good reference for the implementation of one.

Low impact hypervisor support for TEE

It is likely that system developers will want to utilize different hypervisor implementations than the one that gets integrated in Apertis. It is believed that implementing a TEE mediator will be a relatively complex task and system developers may wish to minimize the changes that are required in the hypervisor they are using. In instances where a single VM is expected to access the TEE, we envisage that cut down support could be added to the hypervisor, though more work is required to prove the accuracy of this plan and may be heavily effected by the exact choice of hypervisor.



255

256 This is expected to require the hypervisor to:

- 257 • Only allow one specific guest to call to OP-TEE via SMC and blocking
258 any other guest from making SMC calls.
- 259 • Either translate the kernel physical addresses to *real* physical addresses
260 *or* ensure the Linux kernel can do the translation itself. A minimal imple-
261 mentation of this could be an identity mapping in the translation tables of
262 the guest so that no actual translation is needed (a flat mapping between
263 the intermediate and real physical addresses).
- 264 • Ensure memory areas used for OP-TEE communications are pinned. A
265 minimal implementation of this could be for the hypervisor to carve out
266 a specific region of physical memory for the guest that it will not touch
267 leaving the guest kernel in full control of said memory (allowing it to do
268 the pinning).

269 One optional, but probably desirable, requirement for the hypervisor would be
270 for it to validate the memory regions passed to OP-TEE. This would be required
271 to prevent the guest using OP-TEE to access the memory of other guests (or
272 even the hypervisor) via OP-TEE.

273 Enabling TEE in Apertis

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

277 It is expected that the above OP-TEE support would be integrated into Apertis
278 in a number of phases:

Phase	Description
0	Suitable reference platform selection
1	Integration of core components and basic operation
2	Addition of hypervisor support using upstream supported hypervisor (Xen)
3	Creation of TEE Proxy
4	Further investigation of TEE tunnel and documenting of the process

Reference Platform Selection

A critical part of integrating the OP-TEE functionality into Apertis is providing a working implementation on a reference platform to validate the integrated components and serve as an example for Apertis users. This enables them to experiment with and learn about the capabilities, so that they can implement OP-TEE successfully in their own systems.

To do this Apertis needs a reference platform that provides existing open source support, or one which would require minimal work to implement the required support. In this instance we need a platform that supports OP-TEE and the chosen hypervisor. In order for the TEE environment to be truly useful, it is necessary that guarantees can be made that the boot process hasn't been tampered with. As a result, a platform where we can also integrate secure boot effectively will be valuable.

The existing Apertis reference platforms do not fully meet these requirements. The OP-TEE project is specifically targeted towards the ARM ecosystem and particularly those that provide ARM TrustZone. ARM TrustZone has been improved in later iterations of the technology and standardized with a reference implementation, available as part of the ATF project, for using TEEs. We recommend that a platform that is capable of utilizing ATF is chosen for this reference. An advantage of implementing the TEE using ATF is that this provides a standardized interface for the trusted OS and thus allows Apertis to potentially be used with alternative trusted OS implementations. Whilst the Renesas R-Car platform, an existing reference platform, [appears to have OP-TEE support](#)²⁰, it is not openly available and therefore not viable for Apertis to use as a reference for this functionality.

There are a limited number of ARM based processors, and thus development boards, that are [listed as having Xen support](#)²¹. Two platforms that stand out as potential options at this point are the [96 Boards HiKey960](#)²² and a board based on the Rockchip RK3399.

²⁰<https://optee.readthedocs.io/en/latest/general/platforms.html>

²¹https://wiki.xenproject.org/wiki/Xen_ARM_with_Virtualization_Extensions#Hardware

²²<https://www.96boards.org/product/hikey960/>

308 Core components

309 The following core components would need integrating or work in order to provide
310 basic operation of OP-TEE.

311 Secure Boot

312 Secure boot provides an initial important step in initialization of the TEE by
313 ensuring that the initialization process is able to proceed without interference.
314 Unfortunately this fundamental step is very platform dependent and can not
315 be solved as a general case. Apertis has already taken steps to [document and](#)
316 [demonstrate secure boot](#)²³. At the moment, Apertis only ships some support
317 for secure on the SABRE Lite platform. This provides a good reference for the
318 overall process but, unfortunately, the SABRE Lite is not a good choice as a
319 technology demonstrator for TEE due to its age.

320 We advise the implementation of a TEE demonstrator on a more modern platform,
321 utilizing ATF, to take advantage of the more advanced functionality found
322 in such platforms.

323 In addition to the board verifying the initial binaries that are executed, it is
324 important that the verification of binaries continues through the boot process
325 in order to build a [chain of trust](#)²⁴ so that later stages can determine whether
326 boot was carried out appropriately.

327 ARM Trusted Firmware

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

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

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

340 **Requirements** In order to implement [Trusted Board Boot](#)²⁵ it will be necessary
341 to upgrade `mbedtls`. This functionality is likely to be considered critical by
342 project developers.

²³<https://apertis-website-0b3586.pages.apertis.org/architecture/platform/secure-boot/>

²⁴https://en.wikipedia.org/wiki/Chain_of_trust

²⁵<https://trustedfirmware-a.readthedocs.io/en/latest/design/trusted-board-boot.html>

343 **OP-TEE OS**

344 The OP-TEE project provides the **OP-TEE OS**²⁶ as the trusted OS that runs
345 in the TEE. `optee-os` is packaged for **Debian**²⁷ and **Apertis**²⁸. Like ATF, an
346 Apertis key will need to be used to sign the binaries intended for the TEE
347 to ensure the chain of trust. Currently when OP-TEE is built, it embeds the
348 public key that will be used for verifying TAs. As with the key/keys used in
349 other steps of this process, in order to ensure that products are properly secured,
350 it would be necessary for product teams to at a minimum replace the key used
351 with a product specific one. A product team may wish to modify OP-TEE to
352 support alternative key management solutions, this is **expected by the OP-TEE**
353 **developers**²⁹.

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

358 **Linux Kernel**

359 Debian (and thus the Apertis configuration) does already enable the TEE sub-
360 system on arm64 where ATF can be used. It is understood that this should be
361 sufficient and thus no extra modifications to the kernel will be required.

362 **OP-TEE Suppliment and User Space Libraries**

363 In addition to the trusted OS, the OP-TEE project provides the **OP-TEE sup-**
364 **pliment and TEE Client API**³⁰. The suppliment provides services to OP-TEE
365 that it does not directly provide itself and the TEE Client API provides a user
366 space API in the REE to communicate with the TEE. `optee-client` is packaged
367 for **Debian**³¹ and **Apertis**³². As these components run in the REE they don't
368 need to be signed.

369 **Sample TAs**

370 The **example TAs**³³ should be packaged so that they can be easily installed on
371 an Apertis. This will enable early investigation of TEEs on Apertis, enabling
372 developers to gain experience with using the TEE. It will provide an example of
373 how to best package TAs for use on Apertis based systems. Depending on the
374 task that developers wish to solve using the TEE, these examples may either

²⁶https://github.com/OP-TEE/optee_os

²⁷<https://tracker.debian.org/pkg/optee-os>

²⁸<https://gitlab.apertis.org/pkg/optee-os>

²⁹https://github.com/OP-TEE/optee_os/issues/2233#issuecomment-379253182

³⁰https://github.com/OP-TEE/optee_client

³¹<https://tracker.debian.org/pkg/optee-client>

³²<https://gitlab.apertis.org/pkg/optee-client>

³³https://github.com/linaro-swg/optee_examples

375 fulfill or provide a framework for development of the TEE requirements. For
376 example, the examples show how to implement secure storage, implement tee-
377 supplicant plugins and use a TA to perform encryption and decryption.

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

381 Debos Scripting

382 Once components are added to the Apertis project, we need a way to combine
383 them into an image that can be booted on the target platform. In Apertis
384 this is performed by Debos using configuration files to determine exactly what
385 packages are added to each image. This also allows for the images to be built
386 automatically and regularly using the latest versions of packages. A special
387 image to automate configuration of the boot process can also be generated like
388 the one provided to update the U-Boot bootloader for the [i.MX6 SABRE Lite](#)
389 [board](#)³⁴.

390 Hypervisor Integration

391 Hypervisors come in a number of different types, the main 2 classifications of
392 hypervisor are called “type 1” and “type 2”. Type 1 hypervisors run on bare
393 metal, where as type 2 run on top of a host operating system.

394 Apertis supports the use of [VirtualBox](#)³⁵ for running the Apertis SDK, however
395 this is not intended (nor possible in many instances) to be run on target hard-
396 ware and is very much a type 2 hypervisor. Apertis also has the Linux KVM
397 enabled which turns the Linux kernel into a hypervisor. KVM is not a clear cut
398 type 1 or type 2 hypervisor.

399 We’d expect a product to utilize a type 1 hypervisor such as the open source
400 Xen hypervisor, which contains the experimental TEE mediator. We therefore
401 suggest utilizing this in Apertis to avoid needing to implement TEE mediator
402 support in a different hypervisor at this point.

403 Xen Hypervisor

404 The Xen hypervisor is a GPL-2 licensed type 1 hypervisor. It is supported on
405 Intel and ARM architectures and has the experimental TEE mediator. Xen is
406 packaged for Debian on the `amd64`, `arm64` and `armhf` architectures which will ease
407 adding support to Apertis, though some work is expected to ensure that the
408 version of Xen and the version of the kernel used in Apertis are compatible.

³⁴<https://gitlab.apertis.org/infrastructure/apertis-image-recipes/-/blob/apertis/v2021dev2/mx6qsabrelite-uboot-installer.yaml>

³⁵<https://apertis-website-0b3586.pages.apertis.org/guides/sdk/virtualbox/>

409 As previously mentioned, the TEE mediator support is still considered experi-
410 mental. It is understood that this functionality has been tested on the Renesas
411 R-Car H3 platform. Unfortunately this platform requires components that aren'
412 t openly available and thus does not present a good reference platform for Aper-
413 tis. Some effort may be required to port this functionality to another platform.
414 If there is a wish to support hardware access, significant effort may be required
415 to move the state of art forward.

416 **Linux Kernel**

417 It will be necessary to ensure that the required support is enabled in the Apertis
418 kernel builds, however this should be a relatively straight forward task as the
419 required configuration options are [documented on the Xen website](#)³⁶.

420 **Dual configuration build of OP-TEE**

421 Enabling the virtualization support in OP-TEE results in a version of OP-TEE
422 that only works in virtualized environments. In order to continue to support
423 simple and more complex configurations using a hypervisor it will be necessary
424 to build and package at least two versions of OP-TEE, one supporting virtual-
425 ization and one that doesn't.

426 **TEE Proxy**

427 The concept of a TEE Proxy appears to be new and no existing such projects
428 have yet been found. The implementation of the TEE Proxy will need careful
429 consideration of the potential security implications. The environment in which
430 the TEE Proxy executes will also need to be trusted and will need to be care-
431 fully secured to minimize the risk that third parties could gain access to the
432 communications between the TEE Proxy and it's client or the TEE Proxy and
433 the TEE it's self. The TEE Proxy will also need to be protected against direct
434 attacks against the proxy it's self.

435 **Low Impact Hypervisor Support**

436 The main deliverable from the completion of the low impact hypervisor support
437 effort should be documentation describing in as generic a way as possible the
438 minimal support required from a hypervisor to enable TEE support. This will
439 require a good understanding of the underlying mechanisms used by hypervisors
440 as well as communication between the TEE and REE. We would expect this
441 to be carried out after basic and Xen hypervisor support had been added to
442 Apertis, providing the Apertis team with some of the required experience to
443 formulate this documentation.

³⁶https://wiki.xenproject.org/wiki/Mainline_Linux_Kernel_Configs

444 In order to ensure that this document is accurate and the requirements well
445 under understood, it will be necessary to implement the low impact hypervisor
446 support in a hypervisor. This could be achieved by using Xen or by implement-
447 ing such support for the KVM hypervisor.

448 Test Integration

449 The availability of a [test suite](#)³⁷ test suite provides some coverage of the OP-TEE
450 functionality with minimal effort as this should be usable from automated test-
451 ing. The test suite should also enable developers to easily gain some confidence
452 that OP-TEE was installed and initialized correctly. Whilst not something that
453 the Apertis project will utilize (due to it being proprietary), the availability
454 of an [extended test suite](#)³⁸, which can be purchased from GlobalPlatform may
455 be something that users of Apertis may wish to consider to provide extended
456 testing.

457 In addition to the test suite, OP-TEE provides a [benchmark framework](#)³⁹, which
458 may be beneficial to product teams with a desire or need to track execution
459 performance to ensure that product requirements are being met.

460 Whilst the test suite will test operation of OP-TEE itself, an important part of
461 initializing a TEE is the platform specific secure boot. Unless using a platform
462 very closely aligned with an Apertis reference platform, this step will be the
463 responsibility of the product team.

464 To ensure that this is properly implemented, tests could be developed that
465 attempt to utilize incorrectly signed binaries at the different stages of the boot
466 process to ensure that each step is properly validated, providing a reference for
467 how to test secure boot.

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

³⁷https://github.com/OP-TEE/optee_test

³⁸https://optee.readthedocs.io/en/latest/building/gits/optee_test.html#extended-test-globalplatform-tests

³⁹https://optee.readthedocs.io/en/latest/building/gits/optee_benchmark.html