Developing Secure Services for IoT with OP-TEE: A First Look at Performance and Usability

Christian Göttel, Pascal Felber, Valerio Schiavoni

Developing Secure Services for IoT with OP-TEE: A First Look at Performance and Usability

Christian Göttel[0000−0002−4465−6197], Pascal Felber[0000−0003−1574−6721], Valerio Schiavoni[0000−0003−1493−6603]

University of Neuchâtel, Neuchâtel, Switzerland, first.last@unine.ch

Abstract. The implementation, deployment and testing of secure services for Internet of Things devices is nowadays still at an early stage. Several frameworks have recently emerged to help developers realize such services, abstracting the complexity of the many types of underlying hardware platforms and software libraries. Assessing the performance and usability of a given framework remains challenging, as they are largely influenced by the application and workload considered, as well as the target hardware. Since 15 years, ARM processors are providing support for TRUSTZONE, a set of security instructions that realize a trusted execution environment inside the processor. OP-TEE is a free-software framework to implement trusted applications and services for TRUSTZONE. In this short paper we show how one can leverage OP-TEE for implementing a secure service (*i.e.*, a key-value store). We deploy and evaluate the performance of this trusted service on common Raspberry Pi hardware platforms.
We report our experimental results with the data store and also compare it against OP-TEE's built-in secure storage.

Keywords: OP-TEE· ARM TRUSTZONE· secure storage · IoT

1 Introduction

Despite the availability of security-oriented instruction sets in consumer-grade processors, high-level frameworks that can help developers use such extensions are still at an early stage. Moreover, little has been said regarding the performance and usability of these frameworks. This is unfortunate given that the large majority of devices featuring ARM processors (mobile and not) feature the TRUSTZONE extensions, introduced since 15 years [12], and are constantly being improved with new processor revisions. For instance, ARM recently [4] updated its ARMv8.4 architecture of application processors enabling virtualization in the secure world. The introduction of virtualization in the secure world better improves the isolation of components and resources, and it is expected to boost the trusted applications (TA) ecosystem in developing and using common standards and APIs.

It is only very recently that the first open-source tools aiming to exploit these capabilities have emerged. Notable examples include Linaro ARM Trusted Firmware [14], ARM GNU Toolchain [1], Android's Trusty [11], Trustonic's Kinibi [21], NVIDIA's TLK [18] and finally Linaro's OP-TEE [17].

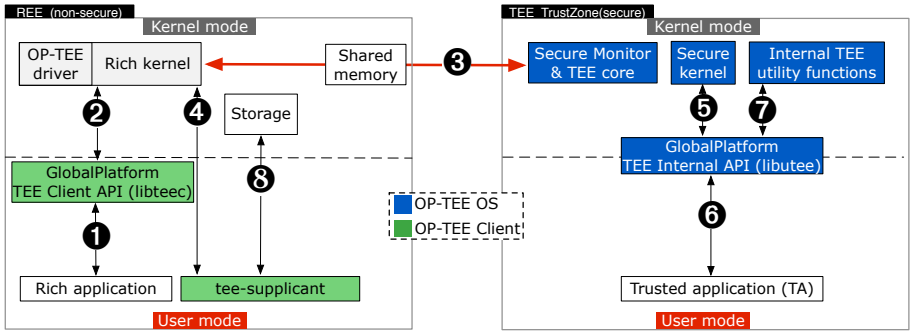


Fig. 1: Organization of components within TRUSTZONE and interaction with OP-TEE

A major challenge for developers of trusted applications resides in the complexity of the secure platforms themselves. Despite the existence of standards and APIs, trusted applications remain OS-specific because of the custom libraries provided by the different vendors. Theses libraries are specialized for the various processors and are required to access secure storage and processing elements. They rely on drivers shipped with the hardware by the silicon manufacturer. Furthermore, dispatching trusted OSs requires trusted OS-specific code in the firmware, which adds up to the issue. This greatly hinders the portability of trusted applications across different trusted OSs and, as consequence, forces TA developers toward implementing and supporting several versions of trusted OS-specific TAs.

In this paper, we focus on a specific framework, OP-TEE [17], which has gained much attention recently and is arguably the most mature open-source framework for developing trusted application with ARM's TRUSTZONE extensions. We describe its architecture and features, and we evaluate its usability and performance by developing a simple key-value store. Such a key-value store could be used to implement a secure password manager, or a secure session manager protecting session data. We also execute OP-TEE's secure storage benchmark and report our results. This preliminary study brings insights into the benefits of such framework, and in particular how it is able to hide the complexity of the underlying vendor-specific libraries and processor, as well as their performance and overhead.

2 Background

2.1 TrustZone in a Nutshell

The TRUSTZONE technology is available in ARM processors since 2003 [4]. It is a hardware-enforced mechanism isolating a *secure world* (trusted) from a *normal world* (untrusted), which includes all components within the SoC as well as peripherals. Thus, TRUSTZONE provides secure endpoints to peripherals on the bus and enables device root-of-trust. Software running in the normal world is unable to directly access secure components and resources. When booting up a TRUSTZONE-enabled SoC, secure firmware is the first software component executed at *exception level 3* (EL3). The secure

Table 1: Comparison of platforms

Device	QEMU	Raspberry
CPU	Intel Xeon E3-1270 v6	Broadcom BCM2837
CPU Frequency	3.8 GHz	1.2 GHz
Memory	63 GiB DDR4	944 MiB LPDDR2
Memory data rate	2400 MT/s	800 MT/s
Disk	Samsung MZ7KM480HMHQ0D3	Transcend micro SDHC UHI-I Premium
Disk Size	480 GB	16 GB
Disk Read Speed	528.33 MB/s	90 MB/s

firmware code is responsible for initializing the platform, installing the trusted *operating system* (OS) and routing secure monitor calls. The trusted OS consists of a small and secure kernel to execute *trusted applications* (TA). Once the secure world is set up, the normal world OS is booted in parallel to the trusted OS running in the secure world. Worlds can be switched via a software-based *secure monitor* (ARMv8-A) or in hardware (ARMv8-M) [3]. The secure monitor acts as a gateway and runs at the highest privilege level EL3 [2].

2.2 The GlobalPlatform Specifications for TEEs

The main specifications for secure digital services and devices are published by industry associations [10,20]. In our study, we focus on the GLOBALPLATFORM specifications for TRUSTZONE. A *rich execution environment* (REE) is an execution environment that involves at least one device and all its components or an OS, excluding any trusted or secure component. In contrast, a *trusted execution environment* (TEE) provides a level of security to protect against attacks and secures data access. The TEE executes alongside the REE, but is shielded from it. A trusted application executes inside a TEE and exposes secure services to applications in the REE. *Trusted storage* is a hardware or cryptographically-protected device capable of storing data [9]. Data can be exchanged between an application in the REE and a TA by three types of shared memory: *whole* (an entire memory region and is allocated by the TEE), *partial* (only a a subset of the *whole* with a specified offset), and *temporarily*, for which a memory buffer region allocate by the application in the REE temporarily shared with the TA for the duration of the API call [7].

2.3 The OP-TEE Framework

OP-TEE [17] is a TEE implementation of GLOBALPLATFORM specifications on top of TRUSTZONE. It can be used alongside a Linux-based distribution running in the REE. TAs are single-threaded executables stored inside the REE. Users develop TAs without having to recompile the entire framework. However, OP-TEE does not provide mechanisms to verify the integrity of a TA. TAs, that do not origin from a secure storage, can compromise the integrity or protection of the TEE upon modification. Alternatively,

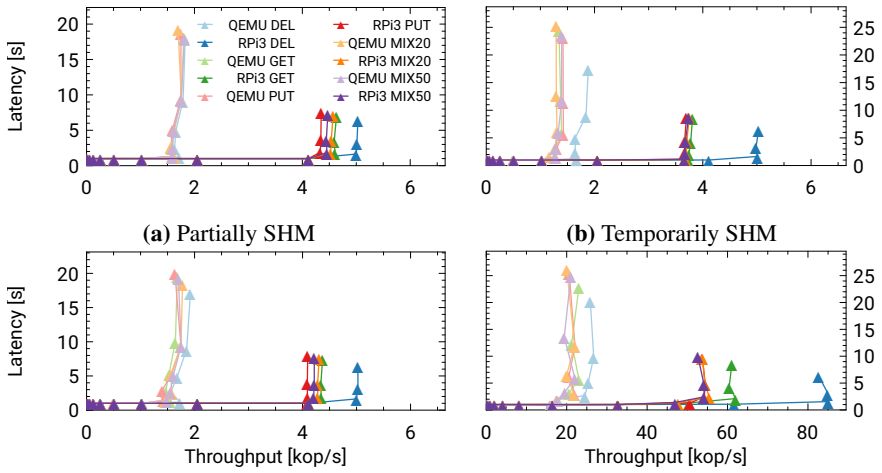
TAs can be directly integrated into OP-TEE as *pseudo TAs*. Pseudo TAs run inside OP-TEE OS' kernel (at secure EL1) as secure privileged-level services without access to GLOBALPLATFORM's Internal Core API. Thus, pseudo TAs can only use OP-TEE's core Internal API.

Secure storage allows applications to offload data from a TA to either the REE file system or a *replay protected memory block* (RPMB) partition of an *embedded multi-media controller* (eMMC) device using the Internal Core API. By default, the OP-TEE OS is configured to use the RPMB [16] if available. The secure storage is accessible and visible only to the TA that created it.

3 Usability

The communication between an application in the normal world and a TA evolves around functions handling the context, session, command and shared memory as shown in Figure 1. This facilitates interoperability between different GLOBALPLATFORM API compatible TEE implementations and allows REE applications to set up multiple contexts. A context is initialized by referencing the device file Figure 1-❶ connecting to the TEE driver Figure 1-❷. TAs are identified by a *universally unique identifier* (UUID), which is referred to when setting up a session to a TA Figure 1-❸. To set up a session, OP-TEE will load the TA from the normal world to the secure world with the help of `tee-supplicant` Figure 1-❹. The `tee-supplicant` is a daemon running in the normal world used by OP-TEE to request services from the REE. These steps are skipped when a session to a pseudo TA is established. A TA can initialize and set up its environment upon TA creation and session establishment (Figure 1-❺ & Figure 1-❻). From this point on, the REE application can request services from the TA by invoking commands. These commands can pass up to four parameters, which are either values or references to shared memory regions. Values are pairs of unsigned 32 bit integers. Shared memory regions are allocated, registered and released through GLOBALPLATFORM API calls in `libteec`. Without the availability of `libteec`, developers would have to communicate directly with the kernel driver through `ioctl` system calls.

In OP-TEE, TAs can use services accessible through GLOBALPLATFORM Internal Core API Figure 1-❻ implemented in `libutee`. TAs are statically linked against `libutee`, which wrapps the API functions around assembler macros to OP-TEE OS system calls. The library provides interfaces to secure storage Figure 1-❸, time, arithmetic and cryptographic operations Figure 1-❼. The secure storage API encrypts data objects by the use of a secure storage service. The encryption process involves three keys: *secure storage key* (SSK), *trusted application storage key* (TSK) and *file encryption key*. The SSK is generated from the *hardware unique key* and is used to derive TSKs. Each TA has a TSK that is generated from the SSK and the TA's UUID. Both SSK and TSK are generated using HMAC SHA256 algorithm [16]. Finally, for every created file, a *file encryption key* (FEK) is generated from the pseudo random number generator. The encrypted data objects are then transferred to the `tee-supplicant` by a series of remote procedure calls and stored in a special file. OP-TEE further provides TAs with libraries for TLS and SSL protocols (`libmbedtls` [5]), arithmetic (`libmpa`) and a subset of ISO C functions (`libutils`). These libraries are used in part by OP-TEE



(a) Partially SHM **(b)** Temporarily SHM

(c) Whole SHM **(d)** Inside REE

Fig. 2: Throughput-latency plots of shared memory types for key-value TA in TEE and REE

to implement GLOBALPLATFORM's Internal Core APIs, in particular the *Arithmetical API* and the *Cryptographic Operations API*. Without these libraries, TA developers would have to provide this code, and they would not be able to just simply link their TA's code against this set of initial libraries. Once the REE application has no further service requests, the session is terminated and the context is destroyed.

4 Performance Evaluation

4.1 Setup

The OP-TEE framework has built-in support for QEMU [6] deployments, which provides an easy to use and inexpensive way for developers to explore ARM TRUSTZONE, with little to no downsides compared to hardware deployments. For this reason, we decided to deploy the key-value store and OP-TEE's Sanity Testsuite v3.2.0 [15] on the following two platforms: Dell PowerEdge R330 Server and Raspberry Pi 3B v1.2. The Dell PowerEdge R330 is running Ubuntu 18.04.1 LTS with the 4.15.0-43-generic Linux kernel and is used to emulate the Raspberry Pi 3B platform with QEMU v2.12.0. A comparison of the two platforms can be found in Table 1. OP-TEE provides a build environment which, by default, deploys and emulates its OS on an ARM Virtual Machine `virt` using a Cortex-A57 with no more than two cores. The deployment was changed to match the specification of the Raspberry Pi 3B platform as close as possible.

4.2 Shared Memory

We have ported a simple key-value store to a TA, in order to evaluate the overhead and performance of different types of shared memory. As basis, we use a modified version of the hash table implementation of `kazlib` v1.20 [13], removing support for contexts

and dynamic tables. The hash table is static, uses separate chaining to resolve collisions, applies a modular hashing and has 251 chains. We time every `DEL` (delete), `GET` and `PUT` operation for each benchmark by referring to `CLOCK_MONOTONIC` in the REE. Operations are uniformly distributed and issued 256 times at a rate of 1 to 32 768 operations per second.

When using whole or partially shared memory introduced in Section 2.2, the REE application requests a shared memory region of 512 KiB and fills it with random data from `/dev/urandom`. Similarly, the REE application allocates and initializes a 512 KiB buffer used as temporarily shared memory. Before every invocation of a key-value operation, a random offset into the shared memory region is computed, which is also used as key. A chunk size of 1 KiB beginning at the random offset is used as data object. The `PUT` benchmark starts with an empty hash table. The `DEL` and `GET` benchmarks start with a pre-populated hash table of 256 data objects. Finally, the mixed benchmark (ratio of `GET` and `PUT` operations) begins with a pre-populated hash table relative to the percentage of `GET` operations.

Figure 2 shows throughput and latency for the different shared memory types and for running the key-value store entirely in the REE. On the QEMU platform, the operations do not separate as well as on the Raspberry platform; we assume due to reaching an I/O bound. The operations on the Raspberry platform separate as expected according to their throughput (lowest to highest): `PUT`, `MIX50`, `MIX20`, `GET`, and `DEL`. The overhead of the `PUT` operation is due to memory allocation, memory copy and object insertion. The `GET` operation looks up a data object and copies it to shared memory, resulting in a lower overhead. The higher the portion of `PUT` operations in the `MIX` benchmarks is, the slower the average operation speed becomes. Thus, `MIX50` (50 % `PUT` operations) has a lower average throughput than `MIX20`. The `DEL` operation looks up a data object and frees its memory, avoiding time consuming memory operations. Comparing TEE throughput against REE throughput yields a 12 to 14 × overhead on the QEMU platform and a 12 to 17 × overhead on the Raspberry platform. A similar experiment was conducted in [19], where they compared the time spend in normal and secure world when invoking a noop operation.

4.3 Secure Storage

The secure storage benchmark is part of the OP-TEE sanity test suite adhering to the *Trusted Storage API for Data and Keys* described in [8]. Neither of the platforms is equipped with an eMMC, for which reason the secure storage has to be offloaded to the REE file system. The benchmark executes three commands `WRITE`, `READ`, and `REWRITE`, for data sizes in the range of 256 B to 1 MiB, that are accessed in chunks of at most 1 KiB. The `REWRITE` command first reads data from an object, resets the cursor and writes the data back to the same object. The data to be stored in the secure storage is allocated and filled with scrambled data within the TEE.

Figure 3 shows the overhead of accessing data in chunks of 1 KiB in the secure storage. In general, the overhead becomes more significant with increasing data sizes, more precisely once the data size exceeds the chunk size. Maximum speed is achieved when the data size equals the chunk size. Overall, the `REWRITE` command has the highest overhead, because it basically executes the `READ` and `WRITE` commands in one batch.

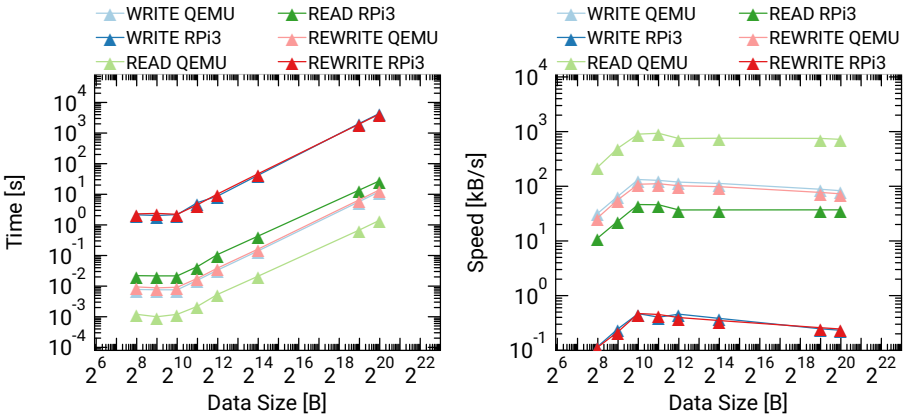


Fig. 3: Secure storage benchmark execution time and throughput

5 Concluding Remarks

Development of secure services benefits from well established APIs and standards. OP-TEE has implemented several of GLOBALPLATFORM's specifications and APIs and provides common interfaces for secure services. We have ported a simple key-value store to a TA and we have studied the performance and usability of secure storage and shared memory. The results of our benchmarks have shown that requesting services from TAs in TRUSTZONE on ARMv8-A using OP-TEE incurs a significant overhead compared to service execution in the normal world. Limiting the space available to a TA is sensible, in order to minimize the trusted computing base. However, the default memory limit of 1 MiB for TAs in OP-TEE becomes a major inconvenience with respect to secure storage and shared memory.

Generating the SSK in OP-TEE requires the HUK. However, most platforms lack of documentation to access or obtain the HUK. OP-TEE avoids this issue by considering a static string value instead of the HUK. This alternative can potentially weaken the cryptographic protection of the objects stored in the REE file system of the secure storage. TEEs would greatly benefit from unrestricted access to HUKs and could so improve the protection of trusted storage.

We expect the trusted application ecosystem to improve portability of TAs among TEEs. Furthermore, we hope that our evaluation of usability and performance of TAs provides deeper insight into future development of trusted services.

Acknowledgments

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the LEGaTO Project (legato-project.eu), grant agreement No 780681.

References

1. Arm Limited: ARM GNU Toolchain. https://developer.arm.com/open-source/gnu-toolchain/gnu-a, Accessed on: 22.02.2019
2. Arm Limited: Fundamentals of ARMv8-A. https://static.docs.arm.com/100878/0100/fundamentals_of_armv8_a_100878_0100_en.pdf (Mar 2017), Accessed on: 22.02.2019
3. Arm Limited: Trustzone technology for the ARMv8-M architecture. https://static.docs.arm.com/100690/0200/armv8m_trustzone_technology_100690_0200.pdf (Mar 2017), Accessed on: 22.02.2019
4. Arm Limited: Isolation using virtualization in the Secure world. https://developer.arm.com/-/media/Files/pdf/Isolation_using_virtualization_in_the_Secure_World_Whitepaper.pdf?revision=c6050170-04b7-4727-8eb3-ee65dc52ded2 (2018), Accessed on: 22.02.2019
5. Arm Limited: mbed TLS. https://tls.mbed.org (Feb 2019), Accessed on: 22.02.2019
6. Bellard, F.: QEMU. https://www.qemu.org (Jan 2019), Accessed on: 22.02.2019
7. GlobalPlatform, Inc.: TEE Client API Specification Version 1.0 (Jul 2010), GPD_SPE_007
8. GlobalPlatform, Inc.: TEE Internal Core API Specification Version 1.2 (Oct 2018), GPD_SPE_010
9. GlobalPlatform, Inc.: TEE System Architecture Version 1.2 (Nov 2018), GPD_SPE_009
10. GlobalPlatform, Inc.: GlobalPlatform Homepage. https://globalplatform.org (Feb 2019), Accessed on: 22.02.2019
11. Google LLC: Android Trusty. https://source.android.com/security/trusty (Feb 2019), Accessed on: 22.02.2019
12. HEXUS.net: ARM Everywhere. https://hexus.net/static/arm-everywhere/, Accessed on: 22.02.2019
13. Kylheku, K.: Kazlib. http://www.kylheku.com/~kaz/kazlib.html (Nov 2000), Accessed on: 22.02.2019
14. Linaro Limited: Linaro Trusted Firmware. https://www.linaro.org/engineering/projects/arm-trusted-firmware/, Accessed on: 22.02.2019
15. Linaro Limited: OP-TEE Sanity Testsuite. https://github.com/OP-TEE/optee_test/tree/3.2.0 (Jun 2018), Accessed on: 22.02.2019
16. Linaro Limited: Secure Storage in OP-TEE. https://github.com/OP-TEE/optee_os/blob/3.2.0/documentation/secure_storage.md (May 2018), Accessed on: 22.02.2019
17. Linaro Limited: Open Portable Trusted Execution Environment. https://www.op-tee.org (Feb 2019), Accessed on: 22.02.2019
18. NVIDIA Corporation: TLK Repository. http://nv-tegra.nvidia.com/gitweb/?p=3rdparty/ote_partner/tlk.git (Oct 2015), Accessed on: 22.02.2019
19. Pettersen, R., Johansen, H.D., Johansen, D.: Secure Edge Computing with ARM TrustZone. In: Ramachandran, M., Muñoz, V.M., Kantere, V., Wills, G., Walters, R., Chang, V. (eds.) Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security. vol. 1, pp. 102–109 (2017). https://doi.org/10.5220/0006308601020109
20. Trusted Computing Group: Trusted Computing Group Homepage. https://trustedcomputinggroup.org (Feb 2019), Accessed on: 22.02.2019
21. Trustonic: Trustonic Kinibi. https://www.trustonic.com/markets/iot (Feb 2019), Accessed on: 22.02.2019