Computer Systems
Science and Engineering

Tech Science Press

DOI: 10.32604/csse.2023.038664
Article

The Trade-Off Between Performance and Security of Virtualized Trusted Execution Environment on Android

Thien-Phuc Doan, Ngoc-Tu Chau, Jungsoo Park and Souhwan Jung*

Soongsil University, Seoul, 06978, Korea
*Corresponding Author: Souhwan Jung. Email: souhwanj@ssu.ac.kr
Received: 23 December 2022; Accepted: 24 February 2023

Abstract: Nowadays, with the significant growth of the mobile market, security issues on the Android Operation System have also become an urgent matter. Trusted execution environment (TEE) technologies are considered an option for satisfying the inviolable property by taking advantage of hardware security. However, for Android, TEE technologies still contain restrictions and limitations. The first issue is that non-original equipment manufacturer developers have limited access to the functionality of hardware-based TEE. Another issue of hardware-based TEE is the cross-platform problem. Since every mobile device supports different TEE vendors, it becomes an obstacle for developers to migrate their trusted applications to other Android devices. A software-based TEE solution is a potential approach that allows developers to customize, package and deliver the product efficiently. Motivated by that idea, this paper introduces a **VTEE** model, a software-based TEE solution, on Android devices. This research contributes to the analysis of the feasibility of using a virtualized TEE on Android devices by considering two metrics: computing performance and security. The experiment shows that the **VTEE** model can host other software-based TEE services and deliver various cryptography TEE functions on the Android environment. The security evaluation shows that adding the **VTEE** model to the existing Android does not add more security issues to the traditional design. Overall, this paper shows applicable solutions to adjust the balance between computing performance and security.

Keywords: Mobile security; trusted execution model; virtualized trusted execution environment; hypervisor

1 Introduction

Nowadays, it is common for people to store personal information and make financial transactions on their mobile devices. Because of that reason, the mobile ecosystem becomes a profitable attraction for hackers and malicious application developers. Such a problem holds a great impact on Android, which is one of the most popular Operating Systems (OS) for mobile. So far, many solutions have been introduced and applied to the Android operating system for leveraging security. Those solutions include software-based solutions such as SEAndroid, Android Binder, and Google Mobile Services (GMS), and hardware-based solutions such as TEE.

The research on securing memory started with a series of inventions by Texas Instrumentation in the 1980s [1,2]. The next invention is *secure execution,* introduced by Nokia, which provides a standard processor with the ability to implement both secure and insecure OSs [3,4]. In 2004, ARM introduced TrustZone as a security system for their architecture [5]. With the use of hardware-enforced security, mobile devices armed with TrustZone can reduce the domain of the attacks to normal applications. Until now, as ARM dominated most of the Android-based mobile market, TrustZone has been widely deployed as a TEE environment for Android devices.

Although hardware-based TEEs such as TrustZone could ensure high-level security, these solutions also highly depend on many different hardware technologies and TEE system providers. Although most Android devices use the ARM architecture, there are still candidates for Intel Atom processors in the market [6]. Because of that, TEE services that were originally developed for TrustZone will not be available on Intel devices. On the other hand, the authors in [7] have provided a categorization of TrustZone-assisted TEE systems in which multiple of them are able to support Android devices. It is also noticeable that most of the TEE systems provided in the paper are proprietary. Since most of the TEE systems are proprietary, it is costly for an Original Equipment Manufacturer (OEM) to deploy their secured apps into Android devices. Different types of TEE platforms and systems also limit OEM developers from providing cross-platform trusted services. In the case of ordinary developers, they have no or limited access to TEE functionality, and there is also no alternative solution for hosting their security services.

Software-based TEE, on the other side, gives more flexibility in deployment, especially mobile device systems. Mobile devices, and their accompanying operating systems, such as Android, are reaching a certain level of complexity where multiple security errors in such software are inevitable. For example, from the beginning of 2021 to October 2021, there have been more than 385 CVEs related to Android, some of which allow privilege escalation, allowing bad guys to control mobile devices remotely without the need to give user permission (indirectly). Rapid delivery of patches is only a temporary solution, while developing solutions that better protect the system is the long-term way. Limited access to and development of security features using hardware-based TEEs is hindering the continued evolution of the protection solution.

This study was conducted with the motivation to develop a new research direction on Android-based virtualized TEE. In order to enhance a non-OEM friendly environment, VTEE was designed with less hardware dependency. The model only requires virtualization-supported hardware (i.e., hypervisor mode enabled), which is available by default in most modern Android chipsets. As far as our best knowledge, this is the first work to implement virtualized TEE environment for the Android platform. VTEE model is designed to be a broker service that responds to TEE service requests. The responses can be returned from either software-based or hardware-based TEEs, depending on the current hardware status. By combining the model with virtualization technology, VTEE can provide an isolated environment for hosting another software-based TEE, such as Open-TEE. In this way, VTEE provides much more flexibility for developing new trusted applications on demand, which is hardly done with previous solutions. The contributions for this paper are as follows:

- A broker model named **VTEE** is proposed to solve the availability issue of existing hardware-based TEE that causes a major obstacle in developing security features that take advantage of TEE on the system by combining it with a software-based solution.

- This research also proposes methods to adjust and analyze the trade-off between *performance* and *security* of *software-based* TEE solutions for the Android platform. The demonstration shows that the *computational overhead* and *security* are acceptable while executing the VTEE model on the Android environment.

The paper is organized as follows. Section 2 contains knowledge about TEE architecture and components. The section also includes related works on hardware-based and software-based TEE. Section 3 contains the detailed design of the proposed model. Section 4 provides the evaluation and experiment results of the proposed model, including the overhead, performance, and efficiency of VTEE. Section 5 concludes the paper with future research plans for the VTEE project.

2 Related Works

2.1 Introduction to TEE Architecture

TEE architecture and design have been explained in various documentation [8–10]. A common TEE design contains a normal world (or non-secure, also known as Rich Execution Environment in some documents) and an isolated environment called a secure world. Each of the normal and secure worlds has its operating system and kernel. An application that runs inside a secure world is referred to as a Trusted Application (TA). On the other hand, an application running in the regular operating system is called an Untrusted Application (UA).

The TEE drivers are designed as Linux kernel drivers located under`/dev/`, and the drivers are named differently between TEE platforms. For Qualcomm's Secure Execution Environment (QSEE), the TEE driver is named *qseecom* [11]. Trustonic is another vendor of TrustZone that provided TEE drivers under the name *mobicore* [12]. Other projects such as Portable Trusted Execution Environment (*OpTEE*), *OpenTee*, and *Intel Software Guard Extensions (SGX)* also provided their own Linux drivers [9,13–15]. Trusted execution environments, such as TrustZone and Intel SGX, could gain control of hardware resources and isolate those from the normal world. Because of that design, TEEs are able to prevent attacks from the normal world to the processes that interact with the isolated resources.

2.2 Levels of Security in Android

In this section, this work provides an explanation of security layers in Android where trusted execution regions are placed at the highest security level. Fig. 1 illustrates the four main security layers in Android.

The order of Security Levels (SL) is increasing from the highest layer to the lowest layer (Secure mode). At the application level, protections are made through Android components such as IPC binder and Dalvik/ART runtime. The malicious apps can also be filtered out by GMS.

The next level of security in an Android device is provided from the Operating system (OS) level with Discretionary Access Control (MAC). Based on the document about Linux Security Framework (LSF) written by Wright and others [16], the LSM hook is a module that provides MAC along with Discretionary Access Control (DAC). In normal Linux, there are many options for MAC, such as AppArmor (for Debian distribution), SELinux (for Redhat distribution), and more. For Android, the MAC option is a modified version of SELinux, which is called as SEAndroid. From Android 10, the control group (Cgroup) is also a part of the security layer for controlling the resources of a specific process or thread.

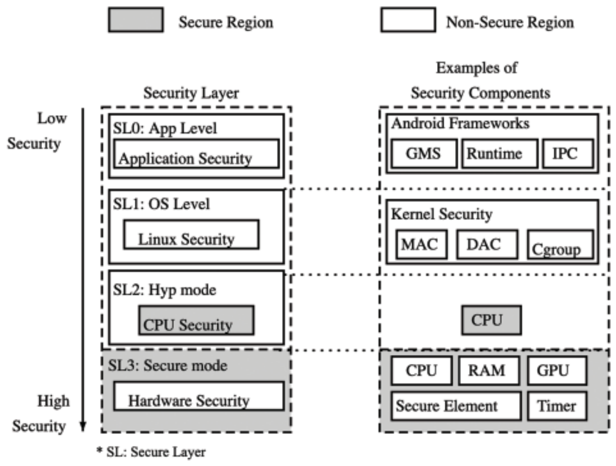


Figure 1: Android security levels

The Hypervisor (HYP) mode is the next privileged level in ARM architecture which is higher than kernel privilege at the OS level. Hyp mode is used for hosting the hypervisor code in a secure CPU state [17]. However, the HYP mode still has a lower security level than the TEE design since it only supports CPU protection. On the other hand, TEE could support various resource protection, including memory, GPU, and other security elements.

2.3 Hardware-Based TEE

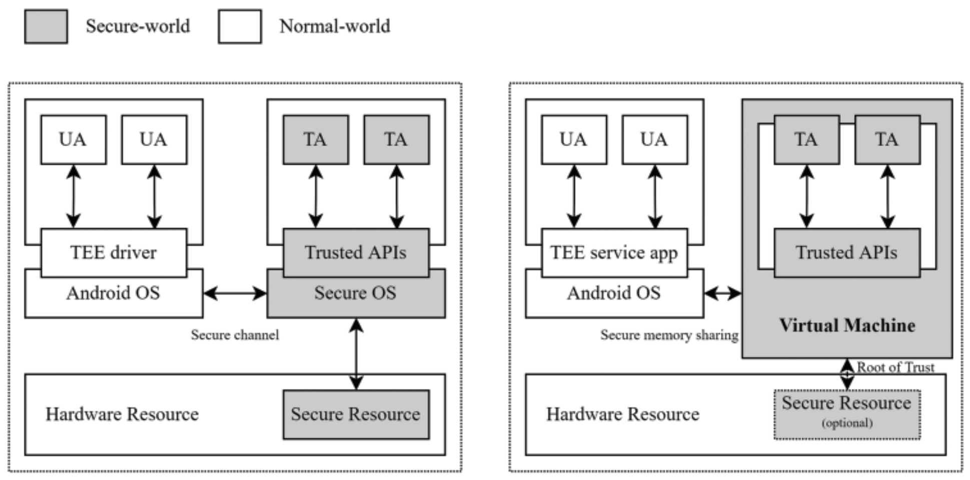
Hardware-based TEEs, as illustrated in Fig. 2a, have been invented by various CPU vendors and deployed in mobile devices for several years. TrustZone, a System-on-Chip (SoC) security solution for securing applications and data, was introduced in 2004 and officially released by ARM in 2005 [18]. In 2016, Intel introduced another hardware-based TEE solution in the name of Intel SGX [15]. As for the open-source community, Open OpTEE is a project developed and supported by Linaro Limited [14]. The project is commonly known since it supports various development boards as well as Quick Emulation (QEMU) environment. Andix OS is another open-source TEE project that mainly serves research purposes on TrustZone [19].

2.4 Software-Based TEE

Software-based solutions is more attentive to solve limitations of hardware-based ones. Firstly, it is cheaper to have software-based solutions deployed because it doesn't depend on specified hardware requirements such as ARM TrustZone, Intel SGX. Secondly, deploying software-based one is easier to add new function when non-OEM developer need to add special security functions to their application.

Software solutions for deploying TEE are mostly built on top of virtualization environments, as shown in Fig. 2b. A virtualizing OS is run by the host system to provide trusted applications for securely computing and processing sensitive data. Previous researches on developing software-based TEE environment could be categorized as follows.

Intra-privilege isolation: known as same-privilege isolation, where the virtualized TEE is located in same security level to the rich world application. Not many studies focus on this approach due to less security provided. Related techniques proposed to enhance isolation for virtualization in ARM platforms [20,21].



(a) Hardware-based TEE (b) Software-based TEE

Figure 2: Hardware-based and software-based TEE architecture

Inter-privilege isolation: there are couple of techniques proposed to deploy isolation environment using ARM platform [22,23]. These techniques leverage the CPU separation of ARM HYP mode to provide isolation. Li et al. [24] proposed a smaller trusted computing base (TCB) size than using KVM in term of line-of-code with the goal of protecting integrity and confidentiality for virtual machines running in cloud platform. Open-TEE which provides a software-based TEE environment followed by the Global Platform Standard [9]. In CCS '19, Zhao et al. introduced SecTEE [25], their Software-based architecture, which was built on top of OP-TEE. SecTEE could resist side-channel attacks self-sufficiently with its software-based enclave architecture. Another approach presented by Lee et al. is SofTEE [26]. Their solution provides a memory isolation space by separating the kernel into the normal and secure worlds, which is inspired by the ARM TrustZone.

In short, software-based TEE environments have been developed for different purposes using various approaches. However, these works didn't show the way to adjust between the security and performance of virtualized TEE solutions. Furthermore, virtualized TEE solutions are mostly implemented for cloud computing but are limited to Android platforms.

3 Motivation and Proposed Model

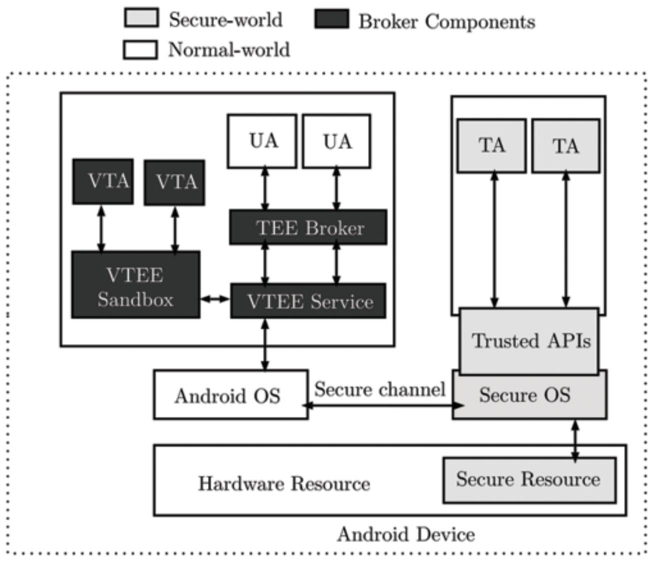
3.1 Motivation

On the one hand, Hardware-based solutions rely on chipset technological support such as TrustZone, SGX, etc., which might lead to the difficulty for OEM developers to provide cross-platform trusted services. On the other hand, current software-based approaches, to our best knowledge, have not supported Android mobile devices, the most popular in the mobile market [27]. Motivated by the existing problems, this study introduces a software-based TEE model called Virtualized TEE (or **VTEE**).

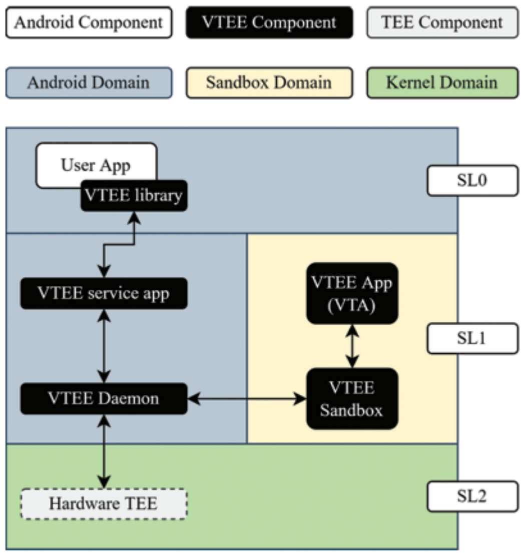
3.2 Proposed Model

This section introduces **VTEE** model with an analysis of the effectiveness of the model for the existing problems of hardware-based TEEs. Fig. 3a illustrates how VTEE can be a part of the Android system as well as the design of the VTEE broker model. The proposed model adds a TEE broker service to the normal world that contains the following components:

- VTEE Library is an Android library (.jar) file for providing an API interface to the VTEE service. This library file is a part of the VTEE broker model.
- VTEE Service App is also part of the VTEE broker model. This app is a system service application in Android that has the same privilege as a system user. This service is responsible for receiving the request from UA and forwarding it to either the original TEE driver or the VTEE sandbox.
- VTEE Sandboxes are software-based TEE solutions that are responsible for providing TEE service for each request forwarded by VTEE Service App. The VTEE sandbox can be designed in HYP mode (as a virtual machine) or at the OS level (as a MAC-protected process). The Fig. 3b illustrates the VTEE sandbox based in OS level mode (SL1).
- VTAs are virtual trusted applications that are responsible to visualized the TEE functionality.



(a) VTEE architecture besides the hardware-based TEE



(b) VTEE sandbox deployed in SL1

Figure 3: VTEE architecture design

The function of this proposed model can be summarized as follows. Firstly, untrusted applications in normal-world requests for secure computation by calling API from the VTEE library. The API helps the application establish, transfer and receive responses from VTEE Service App. Then, the VTEE Service App collects requests and forwards data to VTEE Sandboxes securely. VTEE Sandboxes deploy TA inside an isolated environment, ensured by HYP mode, following the request data from UA. Finally, TEE functions are safely computed in side TAs, and the result will be sent back to UAs via the broker model.

3.3 VTEE Library

The application itself does not have enough and should not have the higher privilege to escape its sandbox. Therefore, a library that provides sound interfaces that acts as a bridge between the application and the VTEE service app. There are two main challenges of deploying VTEE lib: (1) Authentication & authorization; (2) Data transferring.

For authentication & authorization, it is easier whether the private services models are taken place. However, when public services are chosen as the default provider, there is a need for a lightweight CA model, whereas the UA is required a token to access its data or functions provided by the VTEE sandbox. That also ensures that an adversary cannot access other UA data inside the VTEE sandbox.

3.4 VTEE Service App

VTEE service app is a system-privileged application that starts along with the Android OS. VTEE service app binds with the UA using VTEE lib. VTEE service app has full access to the communication channel with the VTEE Sandbox. VTEE service app has a filtering system that ensures UA can get enough required information without breaking the sandbox (e.g., prevent privilege escalation attack). Note that the VTEE service app does not have any encryption module or hold the encryption key. Therefore, the *VTEE service app* only forwards the data to the VTEE sandbox without access to the clear-text data.

3.5 VTEE Sandbox

VTEE sandbox plays the core role in the VTEE Broker Model. It provides TEE services whenever receiving a request from VTEE service. The sandbox contains a request handler (inside), customization of the OPEN-TEE project for providing TEE services, and a lightweight CA system (optional) for authentication while the public service model is set to default. VTEE Sandbox is designed to run at the most privileged level, which can be in HYP mode (L2) or OS (kernel) level (L1). The design depends on the hardware, which PL2 could be considered when hardware-supported virtualization is enabled [28]. In this work, the side-channel attack is out-of-scope. Therefore, the VTEE sandbox is intended to be safe in the highest privilege level.

3.6 Broker Providing

In the proposed model, there are two ways to provide VTEE services: Public service and private service. Fig. 4 illustrates the difference between public and private services. The Public service takes responsibility for providing security services for every application in the system. The Public service should always be on and available for any request forwarded by the VTEE service. The purpose of public service is to reduce the latency which is caused by the initialization of the VTEE sandbox (provided in the Experience part). However, the Public service requires more resources (which is one main concern of the mobile environment) while its VTEE sandbox constantly manipulates a lot of memory disk space during the phone working. Also, security design for *Public service* is more complicated. In the Public service model, a VTEE sandbox is deployed at the Android booting time and is available for serving until Android OS is turned off.

The *Private service* takes responsibility for providing security services for the only corresponding application. Therefore, a VTEE sandbox is only deployed whenever an application requests a set of security services (e.g., in a TEE session). In this way, the Private service requires less security design and fewer resources manipulation. However, the latency from the session opening to the first security service response is much longer due to the VTEE sandbox initialization progress.

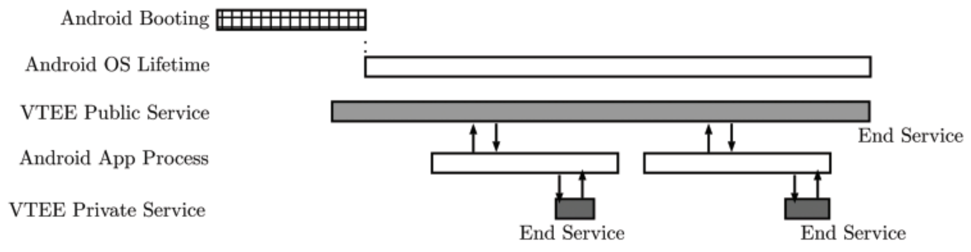


Figure 4: Two ways for VTEE broker to provide: public and private VTEE services

4 Experiments and Discussions of the Proposed Model

While analyzing the proposed model, there are two metrics put into consideration which are performance and security. To prepare for the evaluation, the Hikey960 was chosen as the development board to demonstrate and evaluate the proposed model. Hikey960 is suitable for this research since it is supported by Android Open-Source Project (AOSP). Hikey960 is a reference board to deploy OpTEE [14], a hardware-based TEE OS. In experiments, this work compares VTEE with OpTEE for performance analysis. Furthermore, the board also supports full virtualization on ARM with HYP mode.

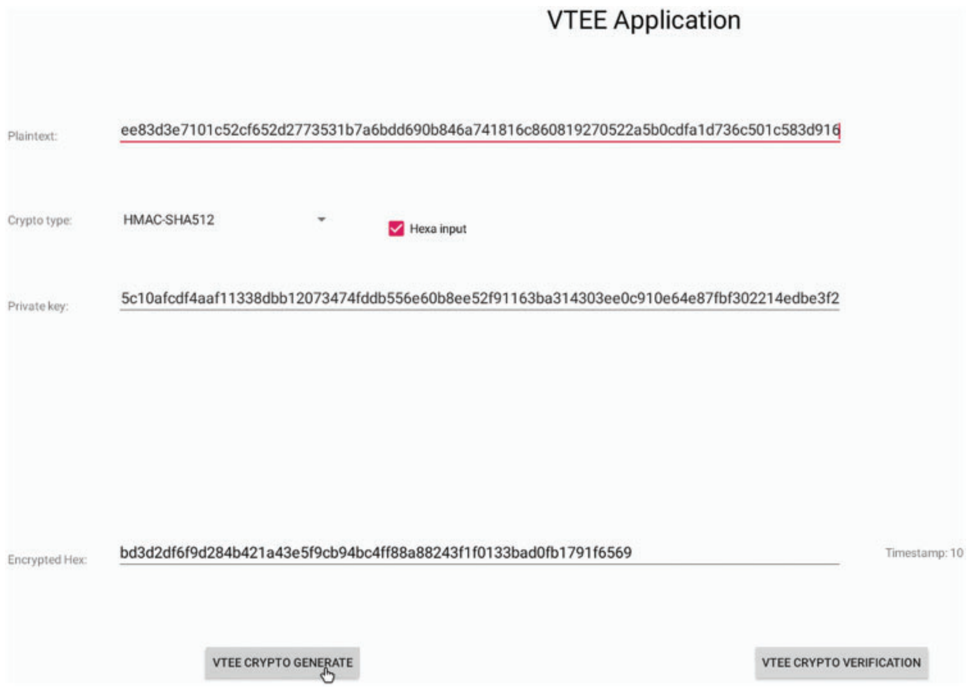
VTEE sandbox can be set up as either a process that depends only on SEAndroid-protected or a Virtual Machine (VM). Although the setup VTEE sandbox process is more lightweight than the SEAndroid-protected version, it is less secure than providing. For that reason, in this paper, the evaluated model was conducted with a VM-based VTEE sandbox. The lkvm-tool was used to deploy the VM environment [29]. LKVM is a project for hosting a VM guest in a lightweight manner than the normal Kernel-based VM solution. With LKVM, it is possible to deploy an arm64-based Debian Linux as TEE OS. In order to provide a lightweight system, there is no hardware virtualization except for virtual IO in the VM.

As for TEE functionality, a custom version of the OpenTEE project was used. The channel between VTEE Service and VTEE sandbox was upgraded with the use of virtualized input/output (virtio). The research also includes a VTEE Android app for measuring the VTEE functionality on Android space (shown in Fig. 5). The testing dataset was collected from the National Institute of Standards and Technology (NIST). For each cryptography algorithm, we randomly choose 20 test cases. Each test case includes an input value (e.g., message to encrypt or decrypt, keys or salt for the hash algorithm) and an expected output value.

4.1 Performance Evaluation: Function Executions

The execution time of each algorithm was measured by the total costs of application requests, application queries, channel communication, and TEE calculation. There are various ways to create a communication channel between VTEE service and VTEE sandboxes, such as ***9p (Plan 9 Filesystem Protocol)*** [30] or shared memory. 9p acts as virtio for file transferring enabled by default in lkvm. ***9p*** allows us to share files between a host and guest VMs. This approach is a passive way to trigger TEE functions and often causes high delay. On the other hand, shared memory is an active way to handle request data and trigger the TEE function. Thus this method can be considered one of the low-delay solutions. Fig. 6a illustrates the performance between running the *Open-TEE* project [9] in a normal environment and inside the VTEE sandbox. The sandbox causes at most **30%** overhead and only a few percent of overhead in some algorithms, such as AES (Advanced Encryption Standard) or

SHA (Secure Hashing Algorithm). It is reasonable since the VM is used HYP mode, a hardware-assist virtualization technology, to support CPU virtualization.


VTEE Application

Plaintext: ee83d3e7101c52cf652d2773531b7a6bdd690b846a741816c860819270522a5b0cdfa1d736c501c583d916

Crypto type: HMAC-SHA512

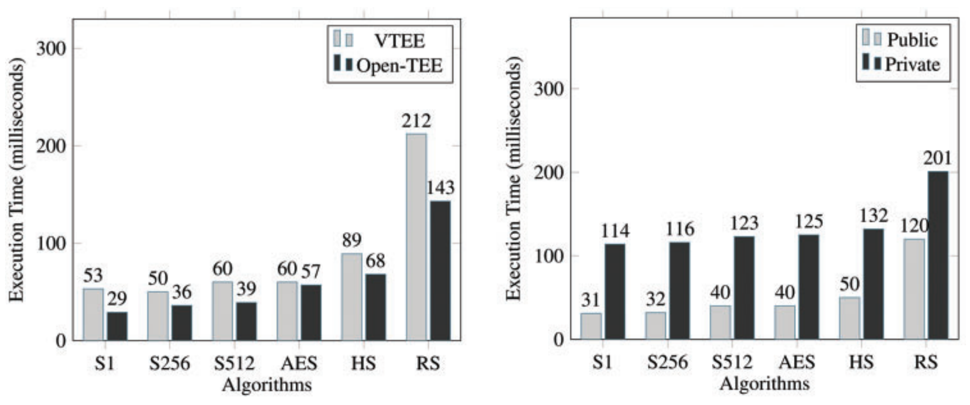
✓ Hexa input

Private key: 5c10afcdf4aaf11338dbb12073474fddb556e60b8ee52f91163ba314303ee0c910e64e87fbf302214edbe3f2

Encrypted Hex: bd3d2df6f9d284b421a43e5f9cb94bc4ff88a88243f1f0133bad0fb1791f6569

Timestamp: 10

Figure 5: Experimental application User Interface (UI). This demo show that the UA requests to compute HMAC-SHA512 from a plain text to VTEE. VTEE sends back results through the secured channel within 10 s



(a) Performance comparison between **VTEE** and **OpenTEE**

(b) Performance comparison between **Public and Private service**

Figure 6: Experimental result in terms of computation overhead

4.2 Performance Evaluation: Services Availability

As introduced in Section 3 there are two services available in our model, which are public and private VTEE services. This experiment measured the execution time between two services. Fig. 6b shows that there is a difference in available time between public and private services. The research has shown that the Open-TEE client will have to wait from 75 to 90 s for the `TEEC_OpenSession` call to work in the virtualized environment. This incident still occurred in public service, and the public TEE service was available at the time of application request because it started at the booting time, which already consumed 90 s of delay time. We also confirmed that the `TEEC_OpenSession` has not occurred in a normal Linux system.

4.3 Performance Evaluation: VTEE and OpTEE

This experiment explored the feasibility of the deployment of VTEE in commercial products, which consider the replacement of hardware security solutions due to the incompatibility and the scalability limitation. A popular hardware-based TEE Operation system (TEE-OS) is OpTEE [14].

Two hikey960s were deployed with the same Android OS version (Android 9.0). The data flow of the test, from the android application to the Trusted application, is deployed in the same way, as shown in Fig. 7. Firstly, when UA requests to use secure functions from the TEE service, it sends data to handle service which runs as Android system service. Handle service, then transfer input data to the running OPTEE/VTEE instance. While *the 9p* protocol is a part of the handle service in VTEE, the VTEE model emulated the same file-sharing function in the OP-TEE version, in which input data is stored inside a sharing file between Android App and either OP-TEE or VTEE. This ensures equality in data transferring between the Android application and the TEE component (OP-TEE and VTEE). After receiving data, OPTEE/VTEE calls to deploy a TA corresponding to the requested function. TA verifies and computes the result and then sends back to OPTEE/VTEE instance as well as the UA.

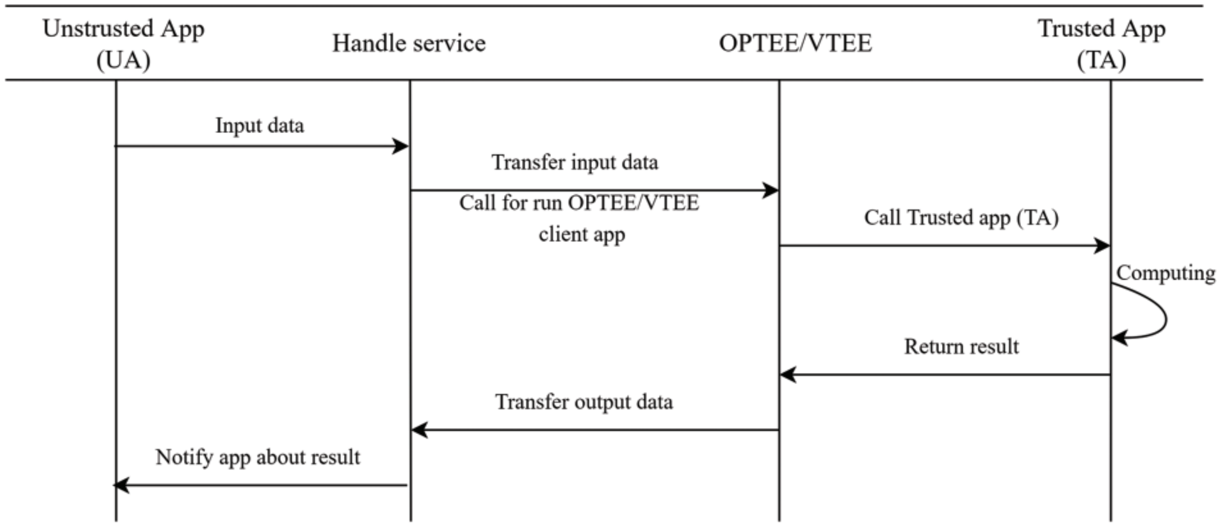


Figure 7: VTEE & OpTEE data flow in the evaluation scenario

Fig. 8 shows the comparison between executing ten algorithms in OP-TEE and VTEE. The results differed from expectations that direct program execution at the hardware (OP-TEE) would be faster. However, most algorithms (except RSA) give the opposite results.

The unexpected performance in software-based TEE is because the number of data transfer points in the OP-TEE is greater than VTEE. Data transfer points are the points that data passes through the process without memory sharing. OP-TEE OS runs in the TrustZone, which requires data transfer from the *Normal world* to the *Secure world* by the UA before being processed in *Trusted Application*

(TA). On the other hand, the VTEE sandbox, which is handled by UA and TA, does not need to switch between two worlds due to the VTEE Sandbox is placed in the same layer with the VTEE service (acts as a data handler). Therefore the total time-consuming:

$$T = T_{Computing} + T_{Data\ transfer}$$

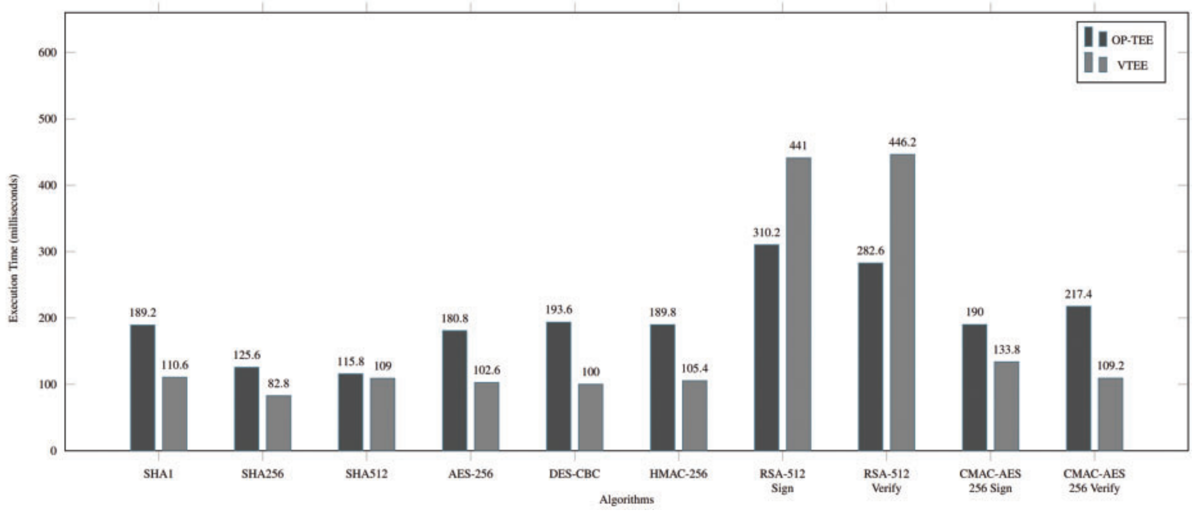


Figure 8: Performance comparison between OpTEE and VTEE

Another reason is that most algorithms witness the faster VTEE except RSA (sign and verify). This result proves that OP-TEE has a shorter computing time than VTEE because RSA requires harder computing power which always performs better in hardware-based solutions rather than software. Other algorithms, which are lightweight computing functions, consume less $T_{Computing}$ than $T_{Data\ transfer}$. The consequence is that the total time-consuming T of VTEE (on average) less than OP-TEE.

The above argumentation supports the high feasibility of delivering VTEE as a Trusted Computing Environment in the wide-range commercial product, which can replace hardware-based solutions from the performance perspective.

4.4 Security Discussion: Limitation of Entirely Software-Based Solutions

According to Fig. 3b provided in Section 3, it is obvious that VTEE sandboxes have less security than traditional hardware-based TEE designs. However, in order to evaluate the security of the proposed model, this research contains various methods for inspecting different Android OS layers. For that, the first version moved the private key of an Android app into the VTEE sandbox. Additionally, another Android app was developed for calculating the crypto algorithm inside.

The result is illustrated in Table 1. The evaluation methods include Dalvik VM (DVM) heap dump, root-based, memory forensic, and side-channel attacks. The first method is DVM heap dump which can be implemented by using the Android-integrated Activity Manager (AM) function. The full memory of a target Android app can be exported to HPROF format (a Heap/CPU profiling tool). The memory of the VTEE Android app and normal Android app can be inspected by this method. However, the private key was not leaked in the VTEE model since the key was stored in the VTEE sandbox. The same result was given from a root-based attack. Because the VTEE VM can be custom with different contexts, it is not possible to attack from root privilege without changing the context.

According to the design of the Android framework, the SELinux context in Android can only be changed by modifying the AOSP source code or by the vendor.

Table 1: Security analysis of VTEE framework

Attack methods	VTEE security checklists					
		VTEE app	VTEE service	VTEE virtual machine	Normal app	TEE app
	DVM heap dump	Heap dump possible by using the activity manager	Injected inside android OS and non-accessible without disabling SEAndroid	Non-accessible without disabling SEAndroid	Heap dump possible by using the activity manager	Non-accessible
	Root attacks	Controllable by root context	Injected inside the android OS and non-controllable by the root user	Controllable by root if SEAndroid is not enforced (not the usual case)	Controllable by root context	Non-accessible
	Memory attacks	Accessible by advanced attacks	Injected inside android OS and non-accessible without disabling SEAndroid	Non-accessible without disabling SEAndroid	Accessible by advanced attacks	Non-accessible
	Side-channel attack	Vulnerable to this attack	Vulnerable to this attack	Vulnerable to this attack	Vulnerable to this attack	Vulnerable to this attack

Another inspection approach is the memory attack, which can be implemented with memory forensic tools and Loadable Kernel Module (LKM) enabled. By satisfying the LKM condition, we were able to disclose VTEE VM memory from known traces. At the time of writing, side-channel attacks are able to exploit all layers of Android. The levels of security in Android OS with the VTEE model after our security evaluations. The table shows that VM-based VTEE sandbox is at the lowest level in untrusted OS. Although the proposed model does not have a large impact on security, it provides an additional layer of isolation and security to the existing Android system.

VTEE, therefore, can play an important role in enhancing the security of Android applications, especially on IoT devices with a lavish hardware-based TEE. For example, cars' smart screen system running Android OS hosts many applications. Even if there are no transactions have been made on the smart screen application, other applications still contain sensitive personal information of the car owner, such as premium accounts, usage caches, and location information for the recommendation system. These data also need to be safe against cyber attacks but require a lower level of security where VTEE is applicable.

The limitation of entirely software-based TEE solutions is inevitable, even though VTEE cooperates with a hardware-based solution to protect the sandbox. However, VTEE could keep the higher security level for the rest. While most side-channel attacks are impacting the VTEE, other resource manipulation attacks are impeded. The trade-off between convenience and security always happens. VTEE model maintains the security perspective in protecting users from more common attacks while delivering more flexibility in TEE function development.

4.5 The Trade-Off Between Performance and Security

As discussed in the previous section, software-based TEE solutions cannot provide as strong security as hardware-based ones. However, software-based solutions such as VTEE allow the deployment of multiple virtualized TAs, running parallelly, isolating from each other and from UAs. Thus, when a system does not require strict security enforcement, software-based solutions are suitable to increase computing power.

In this work, VTEE provides two approaches to strike a balance between security and performance. The first approach is privilege-level isolation. As discussed in Section 3.5, VTEE sandbox can be deployed in HYP mode (L2) or OS (kernel) level (L1). When the VTEE sandbox is run at the OS level (i.e., intra-privilege isolation), the isolation is based on virtualization technology and prior SELinux. On the other hand, deploying the VTEE sandbox in HYP mode provides stronger isolation because it provides CPU protection. In other words, a process needs to be granted to access HYP mode computing resources. As a result, switching between L1 and L2 is costly, but more safety is ensured. The second approach is to move the broker provider from the public to the private model, as discussed in Section 3.6. While private service gives more security (e.g., each UA has a separated TA), the performance decreases more than two times, as shown in Fig. 6b.

5 Conclusion

The interest in solutions that use trusted execution technology has increased in recent years, especially in the mobile environment. Many security threats that affect people directly or indirectly have been discussed over the past decade, continuously improving both hardware-based and software-based solutions. The trusted execution environment is a great improvement in providing a boundary to protect the data as well as the execution of sensitive code. However, existent hardware-based TEE solutions introduce significant effort for ordinary developers to contribute to the real Android production environment. In addition, software-based TEE solutions have not been evaluated for effectiveness in the Andoird environment. This paper presented a work in progress towards implementing VTEE, a Virtualized-based TEE architecture to deliver TEE functionality on Android regardless of hardware support. Leveraging VTEE, this paper provides an analysis of the feasibility of setting up a software-based TEE solution to protect Android devices by considering the trade-off between computing performance and security of the solution. The evaluation results given in this paper have shown insights into the performance and security of the proposed model. VTEE is a promising model for industry products due to its usability, lightness, and openness for developers to extend security functionalities. Besides security and performance, scalability and elasticity could be further evaluated. The fact that VTEE sandbox is a thin virtual machine. Therefore, the design of VTEE could be applied to the edge computing system, which has a huge number of IoT devices in which deploying costs significantly increase. We let this work for future studies.

Acknowledgement: The authors appreciate reviewers and editors for taking the time and effort necessary to help improve the quality of the manuscript.

Funding Statement: This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea Government (MSIT), (No. 2020-0-00952, Development of 5G edge security technology for ensuring 5G+ service stability and availability, 50%) and the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the MSIT (Ministry of Science and ICT), Korea (No. IITP-2022-2020-0-01602, ITRC (Information Technology Research Center) support program, 50%).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] K. M. Guttag, "Secure microprocessor microcomputer with secured memory," 1985. [Online]. Available: https://patents.google.com/patent/US4521853A/en/

[2] K. M. Guttag and S. Nussrallah, "Security bit for designating the security status of information stored in a nonvolatile memory," 1986. [Online]. Available: https://patents.google.com/patent/US4590552A/en

[3] A. Kiiveri and L. Paatero, "Secure execution architecture," 2015. [Online]. Available: https://patents.google.com/patent/US9111097/en

[4] N. Asokan, "Hardware-assisted erusted execution environments: Look back, look ahead," in *Proc. of the 2019 ACM SIGSAC Conf. on Computer and Communications Security (ACM CCS '19)*, New York, NY, USA, pp. 1687–1687, 2019.

[5] T. Alves, "TrustZone: Integrated hardware and software security, white paper," 2019. [Online]. Available: https://ci.nii.ac.jp/naid/10015533638/

[6] Intel, Tablets Powered by Intel, 2019. [Online]. Available: https://www.intel.com/content/www/us/en/products/devices-systems/tablets.html

[7] S. Pinto and N. Santos, "Demystifying arm TrustZone: A comprehensive survey," *ACM Computing Survey*, vol. 51, no. 130, pp. 1–36, 2019.

[8] S. Pinto, D. Oliveira, J. Pereira, N. Cardoso, M. Ekpanyapong *et al.,* "Towards a lightweight embedded virtualization architecture exploiting ARM TrustZone," in *Proc. of IEEE Emerging Technology and Factory Automation (ETFA)*, Barcelona, Spain, pp. 1–4, 2014.

[9] B. McGillion, T. Dettenborn, T. Nyman and N. Asokan, "Open-TEE–an open virtual trusted execution environment," in *Proc. of IEEE Trustcom/BigDataSE/ISPA*, Helsinki, Finland, pp. 400–407, 2015.

[10] A. Machiry, E. Gustafson, C. Spensky, C. Salls, N. Stephens *et al.,*"Boomerang: Exploiting the semantic gap in trusted execution environments," in *Proc. of Network and Distributed System Security Symp. (NDSS)*, San Diego, CA, USA, pp. 1–15, 2017.

[11] S. Makkaveev, "The road to qualcomm TrustZone apps fuzzing," 2019. [Online]. Available: https://research.checkpoint.com/the-road-to-qualcomm-trustzone-appsfuzzing/

[12] Y. Chen, Y. Zhang, Z. Wang and T. Wei, "Downgrade attack on TrustZone," 2017, arXiv:1707.05082. [Online]. Available: http://arxiv.org/abs/1707.05082

[13] Google, TEE documentation, 2015. [Online]. Available: https://kernel.googlesource.com/pub/scm/linux/kernel/git/jkirsher/next-queue/+/devqueue/Documentation/tee.txt

[14] Linaro Limited, OP-TEE website, 2021. [Online]. Available: https://www.op-tee.org/

[15] V. Costan and S. Devadas,"Intel SGX explained," *IACR Cryptol. ePrint Archive*, vol. 2016, pp. 86, 2016.

[16] C. Wright, C. Cowan, S. Smalley, J. Morris and G. Kroah-Hartman, "Linux security module framework," *Ottawa Linux Symposium*, vol. 8032, pp. 6–16, 2002.

[17] C. Dall and J. Nieh, "KVM/ARM: The design ansd implementation of the linux ARM hypervisor," in *Proc. of the 19th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*, ACM Press, Salt Lake City, Utah, USA, pp. 333–348, 2014.

[18] ARM, "ARM security technology: Building a secure system using TrustZone technology," 2009. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

[19] A. Fitzek, F. Achleitner, J. Winter and D. Hein, "The ANDIX research OS—ARM TrustZone meets industrial control systems security," in *Proc. of IEEE 13th Int. Conf. on Industrial Informatics (INDIN)*, Cambridge, UK, pp. 88–93, 2015.

[20] A. Ahmed, N. Peng, S. Jitesh, C. Quan, B. Rohan *et al.,* "Hypervision across worlds: Real-time kernel protection from the ARM TrustZone secure world," in *Proc. of the 2014 ACM SIGSAC Conf. on Computer and Communications Security (ACM CCS' 14)*, Scottsdale Arizona, USA, pp. 90–102, 2014.

[21] Y. Cho, D. Kwon, H. Yi and Y. Paek, "Dynamic virtual address range adjustment for intra-level privilege separation on ARM," in *Proc. of the 24 Annual Network and Distributed System Security Symp. (NDSS 2017)*, San Diego, CA, USA, pp. 1–15, 2017.

[22] C. Dall, S. W. Li, J. T. Lim and J. Nieh, "ARM virtualization: Performance and architectural implications," *ACM SIGOPS Operating Systems Review*, vol. 52, no. 1, pp. 45–56, 2018.

[23] S. Pereira, J. Sousa, S. Pinto, J. Martins and D. Cerdeira, "Bao-enclave: Virtualization-based enclaves for arm," 2022. [Online]. Available: https://arxiv.org/abs/2209.05572

[24] S. W. Li, J. S. Koh and J. Nieh, "Protecting cloud virtual machines from commodity hypervisor and host operating system exploits," in *Proc. of the 28 USENIX Security Symp. (USENIX Security' 19)*, Santa Clara, CA, USA, pp. 1357–1374, 2019.

[25] S. Zhao, Q. Zhang, Y. Qin, W. Feng and D. Feng, "Sectee: A software-based approach to secure enclave architecture using tee," in *Proc. of the 2019 ACM SIGSAC Conf. on Computer and Communications Security (ACM CCS' 19)*, London, UK, pp. 1723–1740, 2019.

[26] U. Lee and C. Park, "Softee: Software-based trusted execution environment for user applications," *IEEE Access*, vol. 8, pp. 121874–121888, 2020.

[27] M. A. Omer, S. R. Zeebaree, M. A. Sadeeq, B. W. Salim, S. X. Mohsin *et al.,* "Efficiency of malware detection in android system: A survey," *Asian Journal of Research in Computer Science*, vol. 7, no. 4, pp. 59–69, 2021.

[28] P. Varanasi and G. Heiser, "Hardware-supported virtualization on arm," in *Proc. of the Second Asia-Pacific Workshop on Systems*, Shanghai, China, pp. 1–5, 2011.

[29] lkvm, lkvm/lkvm, 2019. [Online]. Available: https://github.com/lkvm/lkvm

[30] V. H. Eric and R. Minnich, "Grave robbers from outer space: Using 9p2000 under linux," in *Proc. of USENIX Annual Technical Conf.*, FREENIX Track, pp. 83–94, 2005.