The Dual-Execution-Environment Approach: Analysis and Comparative Evaluation

Mohamed Sabt, Mohammed Achemlal, Abdelmadjid Bouabdallah

The Dual-Execution-Environment Approach: Analysis and Comparative Evaluation

Mohamed Sabt[1,2], Mohammed Achemlal[1,3], and Abdelmadjid Bouabdallah[2]

[1] Orange Labs, 42 rue des coutures, 14066 Caen, France
{mohamed.sabt, mohammed.achemlal}@orange.com
[2] Sorbonne universités, Université de technologie de Compiègne, Heudiasyc, Centre de recherche Royallieu, 60203 Compiègne, France
{mohamed.sabt, madjid.bouabdallah}@hds.utc.fr
[3] Greyc ENSICAEN, 6 Bd Maréchal Juin, 14050 Caen, France

Abstract. The dual-execution-environment approach (dual-EE) is a trusted model that was defined to allow mobile smart devices to guarantee tamper-resistant execution for highly sensitive applications. Although various solutions implementing dual-EE have been proposed in the literature, this model has not been formalized yet. In this paper, we revisit the dual-EE approach and propose a theoretical framework to systematize the design of dual-EE solutions regarding well-established primitives defined in the Multiple Independent Levels of Security (MILS) architecture. We provide a general classification of the different dual-EE proposals based on their isolation properties. We introduce a comparative framework allowing dual-EE solutions to be evaluated across a common set of criteria. The relevance of our framework is examined by applying it on three technologies, each one represents one category in our classification. Results are consistent and explain some hidden and unexpected properties of each technology. For instance, we find that bare-metal hypervisors are ill-adapted to provide high assurance security even though they might improve the overall security level of the system.

Keywords: trusted computing, separation kernel, MILS, TrustZone

1 Introduction

The wide use of modern mobile devices spurs service providers to propose access to their services via smart devices. The growing number of attacks against such devices puts mobile applications under potential security risks. Thus, smart devices are not ideal for services requiring trusted platforms with proved security. Examples include enterprise applications and NFC-based payment solutions. Indeed, the adoption of mobile devices in sensitive business environments has been hindered by the lack of appropriate level of security.

Sensitive-service providers require that their applications run on tamper-resistant execution environment. Such an environment should at least guarantee the following three properties [23]: (1) *authenticity*: the code under execution

should not have been changed; (2) *integrity*: runtime states (e.g. CPU registers, memory and sensitive I/O) should not have been tampered with; and (3) *privacy*: code, data and runtime states should not be observable by unauthorized applications or even underlying OS that might have been compromised.

The default protection mechanisms of smart devices are insufficient to provide tamper-resistant environment. This is due to the fact that these protection mechanisms are mainly based on the operating system, and thus as long as the operating system has not been compromised, sensitive applications are considered as protected. Unfortunately, despite continued efforts to improve the security of operating systems of smart devices [9, 20], they are still essentially untrustworthy for two reasons. First, they are complex and often developed using unsafe languages. Therefore, they are inherently error prone because design flaws and implementation bugs are unavoidable. Second, they allow poor isolation among applications. Indeed, a process with the root privilege can easily access private data and tamper with the execution of other processes. Meanwhile, using specially tailored operating systems can only have very limited success due to their restricted features and compatibility to existing applications.

To remedy this situation, there have been numerous efforts aimed at providing tamper-resistant execution environments. Generally, those efforts can be classified into three categories. First, architectural enhancements based approach, such as XOMOS [19] and AEGIS [24], allows sensitive applications to run on untrustworthy operating system. This approach requires nontrivial modifications to the core processor architecture. Second, micro-kernel based approach, such as SeL4 [16], tries to reduce the trusted computer base (TCB) by running a limited code in the privileged mode. This approach requires a redesign of operating systems, thereby requiring nontrivial modifications to port existing applications. Third, the dual-execution-environment approach (dual-EE), such as TLR [22], solves the problem by multiplexing the feature-rich OS and a specialized OS with restricted functionalities on the same smart device. It relies on the specialized OS to provide tamper-resistant capabilities. Applications that demand tamper-resistant protection run only on the specialized trustworthy OS.

Compared to other approaches, the dual-EE is considered as a promising approach intended for practical use [10]. The literature is full of proposals [2, 10, 13, 17, 27]. However, proposals differ substantially from each other in their design objectives. Some address very specific environments, while others silently seek generic solutions that fit all environments. Too often, authors claim the superiority of their solutions and their assertion is based on self-defined criteria. To make progress, we believe that knowledge regarding the dual-EE approach must be systematized. There is a need to provide a theoretical framework which defines how best to evaluate dual-EE proposals.

In this paper, we analyze the dual-EE approach in the context of the trusted computing domain and the MILS architecture. We propose a standard benchmark and framework allowing dual-EE solutions to be rated across a common, broad spectrum of criteria. Our work provides insights which prove useful in designing more efficient dual-EE schemes. To the best of our knowledge, this is the

first comparative evaluation of the dual-EE solutions available on mobile smart devices. Moreover, we believe that our comparative framework is extendable and sufficiently general to be used to evaluate more fine-grained classifications.

Summary of Contributions: We make the following contributions:

- We construct a compact security model of the dual-EE approach using the *separation kernel* model that provides a relevant abstraction level, thereby contributing to a deeper understanding of the dual-EE approach. We reinterpret well-known security technologies, such as UICC card and TrustZone, in the light of this model.
- We provide a framework to evaluate the dual-EE solutions. Our criteria are divided into three categories: (1) *functional criteria*: schemes are evaluated whether they implement all the requirements of a tamper-resistant environment; (2) *security criteria*: the properties of the separation kernel layer of the scheme are analyzed; and (3) *deployability criteria*: schemes are evaluated whether they could be easily deployed in a real context.
- We provide a classification of the different dual-EE solutions. Nevertheless, our goal is not to provide a comprehensive survey, but to show the relevance and the interest of our abstraction by providing a general classification on which our comparative framework could be applied.

This paper is structured as follows: Section 2 gives a background information on the MILS architecture. In Section 3, we give a general classification of the dual-EE solutions. Section 4 explains our comparison methodology and thoroughly defines our chosen set of criteria. We apply our comparative framework to our classification in Section 5. The resulted comparative evaluation is discussed in Section 6. Section 7 surveys related work, and we end with a brief summary.

2 Background

Building a secure system has traditionally been a cat and mouse game. No sooner are new security mechanisms integrated into systems than hackers find how to bypass them. Research on trusted computing aims to replace this endless game with a methodical process. The domain of trusted computing provides the abstract concepts as well as the theoretical base on which *ideal secure systems* are built [11]. It introduces various security models, called *trusted models*. Each trusted model defines a set of security objectives, a threat model, and security requirements to be satisfied by the component that enforces the security policy. In this paper, we focus on the 'separation kernel' trusted model introduced by John Rushby [21], or more precisely, on MILS [25].

MILS stands for Multiple Independent Levels of Security. This architecture was developed in order to resolve the difficulty to evaluate the assurance level of the widely deployed trusted model 'reference monitor' [18] because of its continually growing complexity. MILS adopts a divide-and-conquer approach. It separates a complex system that includes various modules requiring different levels of security into smaller, hence verifiable components. Thus, instead of

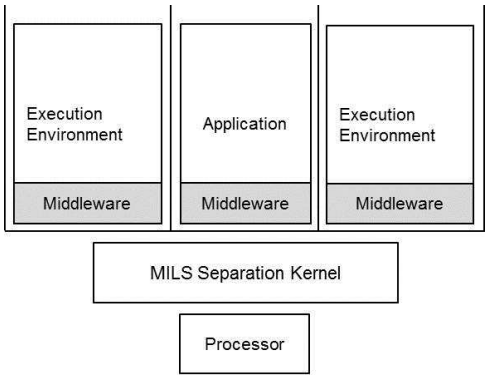

Fig. 1. An Overview of the MILS Architecture

evaluating the whole complex system, these small components are individually evaluated. An abstract view of the MILS architecture is depicted in figure 1. The primary component of MILS is the separation kernel layer (SK). This layer is responsible for creating a set of isolated functional units called *partitions*. All communication between partitions is monitored by the SK layer. MILS is based on separation technology and secure inter-partition communication.

In order to work properly, the SK layer must satisfy several requirements. The SK should be designed so that it cannot be modified or disabled by rogue partitions. In addition, all inter-partition communication requests must go through it. Furthermore, it must be well-structured and small enough, so that its correctness can be validated. In other words, the SK must be (1) tamper-proof, (2) always invoked, and (3) evaluable. These properties correspond respectively to the three principles: isolation, completeness and verifiability.

The dual-EE approach can be seen as a particular case of the MILS architecture where the separation kernel creates two partitions only. The next section provides more details.

3 Dual-EE Solutions

There is an increasing need to use smart mobile devices for applications requiring high security levels, such as enterprise and payment applications. However, their openness and complexity impose fundamental limitations on the security which these devices are able to provide. The dual-EE approach attempts to resolve these limitations by providing trust and high-assurance security while keeping the rich model of smart devices. It brings the best properties of open and trusted systems to smart devices without any compromise. It partitions the system into two execution environments running side-by-side: general-purpose execution environment (GPEE), and secure execution environment (SEE). The GPEE runs the legacy, complex operating system, while the SEE runs a special trusted OS

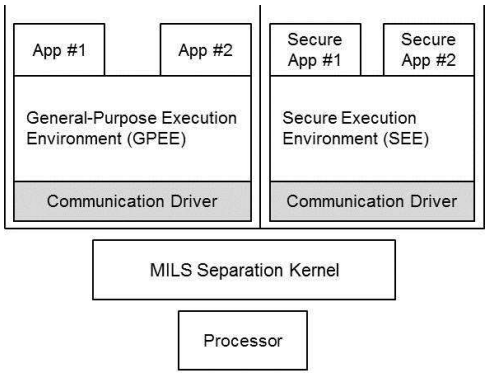

Fig. 2. Representation of the Dual-EE Approach in the MILS Abstraction

with a selection of applications designed specifically for it. The SEE is designed to be trustworthy to provide tamper-resistant capabilities.

Secure isolation is essential for the dual-EE approach. Generally, the security of a system is reduced to that of its most vulnerable component. In dual-EE, the security level is, by definition, supposed to be that of the GPEE. However, the two execution environments are *strongly isolated* so that the compromise of the GPEE does not impact the SEE. Figure 2 depicts the representation of the dual-EE approach in the MILS architecture. In this paper, we consider MILS as the abstract trusted model of the dual-EE approach in which only two partitions exist and the strong isolation is guaranteed by the SK layer. The main advantage of this representation is to use MILS properties as primitives to better understand and thoroughly analyze the dual-EE approach. For instance, MILS defines a set of design principles for the SK layer. These principles provide an abstract model to define the isolation properties required between the two execution environments. We discuss these principles in the next section.

According to their isolation technology, we classify the dual-EE solutions into three categories:

1. **Isolation based on external hardware module:** this category consists in introducing an additional secure coprocessor or integrated circuit to smart devices. A secure coprocessor is a hardware module containing CPU, bootstrap ROM, and secure non-volatile memory. This hardware module is physically shielded from illegal access, and the I/O interface to the module is the only way to access its internal states. Hardware modules cannot only store cryptographic keys without risk of release, but also they can perform arbitrary computations using their CPU. In dual-EE, the SEE runs inside the secure coprocessor. Tamper-resistant execution is guaranteed, since the GPEE and the SEE run on physically two separated memories. Popular examples are UICC card and baseband processor;
2. **Isolation based on bare-metal hypervisor:** this category consists in executing a hypervisor in the most privilege mode of the processor. A *hypervisor*

is a software layer that implements the same instruction-set architecture as the hardware on which it is executed. Thus, it allows multiple operating systems to coexist on the same hardware. Full-virtualization is not possible on ARM processors, which represents 95% of the market of smart devices [1], since ARM is not a virtualizable architecture [7]. ARM introduced hardware virtualization support with the ARMv7 architecture. However, the use of hardware-supported virtualization on ARM is still limited. Instead, para-virtualization approach is prevalent and a myriad of solutions exists [15]. In para-virtualization, OS needs to be modified in order to run on the underlying hypervisor. In dual-EE, the hypervisor plays the role of the SK layer, and the number of virtual machines is limited to two;

3. **Isolation based on special processor extensions:** this category consists in enhancing general-purpose processors with new hardware extensions. These newly-introduced extensions allow the execution of secure code within a potentially compromised OS. The most prevalent secure extensions targeting smart devices is ARM TrustZone [4]. In this paper, we only consider ARM TrustZone because, to the best of our knowledge, it is the most deployed security extensions in practice. A processor with TrustZone extensions provides a special form of virtualization. It enables two virtual processors with two security domains: the "secure" zone and the "normal" zone. In dual-EE, the GPEE resides in the normal zone and the SEE resides in the secure zone. The isolation of both zones or "worlds" is implemented by a complex mechanism using hardware controllers, a configuration bit and a new execution mode called *monitor mode*.

4 Comparison Methodology

In order to evaluate dual-EE solutions, we define three categories of criteria: functional, security and deployability.

4.1 Functional Criteria

Schemes are evaluated whether they implement all the requirements of a tamper-resistant environment. The SEE should provide the following features [12]:

- *Protected Execution*. The execution of secure applications should be protected from any interference caused by malicious software. Runtime states of the SEE should be protected from being observed or tampered with.
- *Sealed Storage*. The integrity, secrecy and freshness of secure applications' content should be protected. Content includes code as well as data.
- *Protected Input*. The SEE should protect their input data from being sniffed or tampered with by malicious applications, such as key loggers.
- *Protected Output*. The integrity and the confidentiality of the output data are protected. Protected input and output do not only concern user interface.
- *Attestation*. The SEE should provide mechanisms allowing secure applications to authenticate themselves to remote trusted parties.

4.2 Security Criteria

In dual-EE, isolation–an essential task to implement–is provided by the SK layer. Schemes are evaluated whether the design principles of the SK layer [3] are implemented in software or hardware in order to ensure:

- *Data Separation*. Data within one partition, namely execution environment, cannot be read or modified by other partitions.
- *Information Flow Control*. Communication between partitions cannot occur unless explicitly permitted by the SK layer.
- *Sanitization*. Shared resources cannot be used to leak information into other partitions.
- *Damage Limitation*. Security breach in one partition cannot spread to other partitions.

4.3 Deployability Criteria

The dual-EE approach is intended to be implemented in a real context. Thus, we evaluate how easy schemes can be deployed. Deployability criteria are numerous. In our study, we only consider the following properties:

- *Support of Legacy Systems*. We evaluate the amount of modifications needed for the GPEE to run on the underlying SK layer. Ideally, no modification, except for the inter-EE communication driver, is required.
- *Cost*. The addition of any software architecture has a cost. We only evaluate the extra silicon cost that the scheme generates. For instance, the addition of hardware module or internal processor extensions are factors which make schemes costly.
- *Overhead*. Schemes should have minimal impact on applications that do not require tamper-resistant protection. They should not incur too much overhead to the SEE either.
- *SEE Performance*. We evaluate how fast the SEE could execute complex operations.

Throughout the paper, for brevity and consistency, each criterion is referred to with an italicized mnemonic title. In our study, we will rate each solution based on its capability to offer the criteria described above. We emphasize that it would be naive to rank dual-EE solutions simply by counting how many criteria each satisfies. Some criteria clearly deserve more weight than others. In this paper, we do not suggest any weights, since providing appropriate weights depend strongly on the specific goal for which the dual-EE solutions are being compared.

5 Comparative Evaluation

We now use our criteria to evaluate three different solutions of the dual-EE approach. Due to space constraints, we only explain one particular solution for each category. We emphasize that, in selecting a particular solution, we do not necessarily endorse it as better than alternatives–merely that it is reasonably representative, or illuminates in some way what the category can achieve.

5.1 External Hardware Module: Smart Card

A smart card is essentially a minimal computing environment composed of a CPU, ROM, EEPROM, RAM, and I/O port. It is capable of running applications (called applets or cardlets) with a high level of security. In smart devices, smart cards come in several flavors. They could be implemented either by an embedded smart card chip, in an SD card that could be inserted in the device, or in the SIM/UICC which is used by mobile operators to authenticate subscribers to their network. In most cases, the SEE consists of Java Card OS, and the GPEE can be any commodity operating system.

Smart cards physically shield the SEE from all types of software attacks coming from the GPEE. Thus, no interference is possible during the execution of secure applications. Moreover, tamper-resistant hardware prevents protected data from being extracted by hardware attacks like microprobing and fault generation. To sum up, smart cards provide *protected execution* and *sealed storage*. *Attestation* is guaranteed, since only authenticated code can run in the SEE. However, smart cards fail to provide *protected input* and *protected output*. In practice, smart cards are designed in a way that there is no direct communication link with the I/O devices. Smart cards, for instance, cannot control user interface to allow users to securely enter their PIN code.

Regarding the SK layer, it is almost implemented in hardware. Both execution environments, namely the GPEE and the SEE, run in two different CPU with their own memory and I/O devices. As a result, the software part of the SK layer does not need to take care of either *data separation* or *sanitization*. However, *damage limitation* depends on how well the inter-EE communication is controlled. The *information flow control* is implemented in the SEE. In fact, the SEE includes the SK part which is responsible for protecting the SEE from accidental or malicious communication attempts that violate the system policy.

For reasons of silicon *cost*, smart cards are often made with limited resources. An additional CPU increases the power consumption and the global cost of the device. Cost and power consumption constraints lead to design smart cards with limited processing power, slow processing speed and small permanent and temporary memory [27]. Therefore, secure applications have low *performance* and cannot perform complex computations. Clearly, smart cards support *legacy systems* and incurs no *overhead* to the SEE.

5.2 Bare-Metal Hypervisor: KVM/ARM

KVM/ARM is *the* ARM hypervisor in the mainline Linux kernel [7]. It is the first hypervisor to leverage ARM hardware virtualization support to run unmodified operating systems on ARM hardware. It builds on KVM and leverages existing infrastructure in the Linux kernel. KVM/ARM is a hosted bare-metal hypervisor, where the hypervisor is integrated with a host kernel. It runs the hypervisor in normal privileged CPU modes to leverage existing OS mechanisms without modification, while at the same time leveraging ARM hardware virtualization. In contrast to standalone bare-metal hypervisors (e.g. Xen), it supports a wide

range of ARM devices despite the fact that there is no standard hardware in the ARM world. In dual-EE, the two execution environments (GPEE and SEE) are two virtual machines running on the underlying hypervisor.

Hypervisors provide isolation properties to prevent potentially malicious VM (GPEE) from attacking another VM (SEE). However, hypervisors only defend against software-based attacks and do not take hardware attacks into account. This isolation property works fine for data centers, but the threat model of mobile smart devices includes hardware attacks. We illustrate the threat by two examples. First, an attacker with physical access to the system can read any data present in memory using the cold boot attack [14]. This attack is based on the fact that RAMs retain their contents for several seconds after power is lost. Second, an attacker with access to the system disk can run a modified version of KVM/ARM that integrates malicious introspection mechanisms to snoop on the runtime states of the SEE. The KVM/ARM hypervisor provides *protected execution*, but not *sealed storage* because encryption keys can be retrieved using, for instance, the cold boot attack. Furthermore, it defines several mechanisms to provide I/O virtualization and interrupt virtualization. Thus, it provides *protected input* and *protected output*. The KVM/ARM alone does not provide *attestation*; trust anchors, such as TPM, are required. It is worth noting that any person who has physical access to the smart device can easily clone the SEE and capture its internal states. This might result in serious attacks, such as rolling back security updates, thus leaving the system vulnerable.

Regarding the SK layer, it is *entirely implemented in software*. All of its design principles are performed by the KVM/ARM hypervisor. Therefore, the hypervisor must be tamper-resistant and evaluable. To the best of our knowledge, KVM/ARM is the smallest bare-metal hypervisor. It is comprised of only 12,883 lines of code. However, it is still too big to be formally verified.

For KVM/ARM, platforms with hardware virtualization capabilities are required. Hardware-based virtualization is not supported on all platforms. Therefore, it presents an additional *cost* to the system. The fact that KVM/ARM leverages hardware virtualization support presents two advantages. First, it can run *legacy systems*, unlike hypervisors based on para-virtualization. Second, the incurred overhead is minimal in comparison with other virtualization solutions. For example, it uses Stage-2 translations to achieve low I/O performance overhead with very little implementation effort. However, KVM/ARM still generates within 10% of *overhead* over a multicore. In some contexts of smart devices, 10% of overhead is not negligible.

5.3 Special Processor Extensions: TrustZone

ARM TrustZone technology can be seen as a special kind of virtualization with hardware support for memory, I/O and interrupt virtualization [4]. This virtualization enables ARM core to provide an abstraction of two virtual cores (VCPUs): secure VCPU and non-secure VCPU. The monitor is seen as a minimal hypervisor whose main role is the control of information flow between the two virtual cores. In dual-EE, the SEE runs on the secure VCPU, while the

GPEE runs on the non-secure VCPU. It is worth mentioning that ARM TrustZone was designed and optimized to implement the dual-EE approach. Indeed, it implements all the hardware extensions defined in [3] and which the SK layer requires in order to work properly.

Similar to bare-metal hypervisor, ARM TrustZone provides *protected execution*, *protected input* and *protected output*, but it does not provide *sealed storage* or *attestation*. However, TrustZone is often completed with additional features, such as secure boot and root of trust (RoT) hardware module, which allow TrustZone to satisfy all the requirements of a tamper-resistant environment.

Regarding the SK layer, it is *mainly implemented in hardware*. The software components to be trusted are minimal, hence *evaluable*. For instance, most CPU registers are banked. Thus, saving and restoring CPU registers are performed by the processor. In addition, TrustZone enables the co-existence of cache entries of both SEE and GPEE. Thus, cleaning the cache memory during a context switch is not required. As a result, the *sanitization* process performed during a context switch is both fast and secure, since it is almost done by the hardware. *Data flow* is well controlled. To enter the secure world, only a well-defined set of interfaces exist. Any transition between the two worlds must go through the monitor mode. This allows the SK layer to satisfy the *completeness* engineering principle.

TrustZone incurs a limited execution *overhead*. The performance is nearly native because both execution environments can access their corresponding resources directly without going through an abstraction layer. Moreover, it can run *legacy systems* without modifications, since each world has its own user and privileged modes, and thereby removing the necessity of instruction emulation. It is true that TrustZone presents an additional *cost* as it requires some modifications to the core processor, but these modifications are already extensively deployed and implemented in a wide range of ARM platforms.

6 Discussion

A summary of our comparative evaluation is presented in table 1. We note that the size of the SK layer is directly proportional to the number of the isolation properties implemented in software [11]. A small SK is better for security because the property of verifiability cannot be satisfied when the SK layer is too complex. Therefore, solutions with many isolation properties provided by hardware are considered *better* than those implementing their SK layer in software.

To our surprise, bare-metal hypervisors achieve the lowest score in our framework. We did not expect this result, since the literature is abundant of solutions presenting hypervisors as a promising approach to improve system security [7, 13, 15]. In this paper, we showed that this approach inherently suffers from three main shortcomings. First, hypervisors come from the world of data centers, and therefore their threat model does not include stolen devices. Even simple physical attacks, like cold boot attacks, can compromise the privacy requirement of tamper-resistant execution. Second, the isolation properties are entirely imple-

Comparison Category	**Comparison Criteria**	*Smart Card*	*KVM*	*TrustZone*
Security Requirements	Protected Execution	√	√	√
	Sealed Storage	√	×	√*
	Protected Input	×	√	√
	Protected Output	×	√	√
	Attestation	√	×	√*
Isolation Properties	Data Separation	HW	SW	HW
	Information Flow Control	SW	SW	HW
	Sanitization	HW	SW	HW/SW
	Damage Limitation	HW	SW	HW
Deployability Criteria	Legacy Systems	√	√	√
	Low Overhead	√	×	√
	Low Cost	×	×	√*
	High Performance	×	√	√

√: satisfies the criterion; **×:** does not satisfy the criterion;
√*: needs widely available additional hardware modules to satisfy the criterion;
HW: satisfied by hardware module; **SW:** satisfied by software implementation.

Table 1. Summary of our Comparative Evaluation of Dual-EE Solutions

mented in software, thereby negatively impacting the verifiability characteristic of the SK layer. Third, although dedicating the whole virtualization layer to hosting security tools present numerous advantages, it is not practical because it will deprive the system from using other virtualization capabilities. Furthermore, it is true that hardware-based virtualization solutions produce better overhead and fewer modifications to existing systems compared to para-virtualization solutions. However, they require specific extensions that are not supported on all platforms. For example, the widely-used Qualcomm Snapdragon MSM8974 and APQ8084 processors do not implement the hypervisor extension.

On the contrary, external hardware modules achieve the highest score in terms of security. Our results are expected, as these modules provide a confined execution environment which protects the application's authenticity, integrity and privacy against even sophisticated physical attacks. Nevertheless, external hardware modules do not fit to a certain kind of secure applications that need user interaction and better processing speed.

As for ARM TrustZone, it comes close to perfect score. Our results are consistent and expected because TrustZone implements all the hardware extensions that the SK layer requires in order to work properly. TrustZone provides a balanced trade-off between bare-metal hypervisors and external hardware modules. Indeed, it does not resist against some physical attacks and it requires a part of the SK to be implemented in software [4]. In addition, TrustZone does not provide sealed storage and attestation without additional hardware modules. However, it is more secure than solutions based on bare-metal hypervisors and more flexible than those based on external hardware modules. Our framework shows that TrustZone-based solutions are efficient for real contexts. Once again,

our results are consistent with existing work. At present, millions of devices integrate TrustZone-based technologies. Examples are ObC in Lumia phones [17], TIMA/TZ-RKP in Samsung smartphones [6], and <t-base of Trustonic [2].

7 Related Work

The two main research directions that our work targets is trustworthy execution and trusted computing in mobile smart devices. Extensive discussion of trusted computing solutions for mobile devices is found in [5]. Authors in [26] evaluate existing hardware security features available on mobile devices for creating tamper-resistant execution. However, these surveys fail to identify dual-EE as a promising model that brings trusted computing for smart devices.

Earlier works focus solely on a particular dual-EE technology discussing the advantages that it presents compared to other existing technologies. For instance, the case of TrustZone is presented in [27] and that of bare-metal hypervisors is presented in [13]. Too often, authors assert the superiority of their solution without explicitly stating their evaluation criteria. As such, consensus is unlikely as objective comparison between different solutions is not possible.

The closest work to ours is [8], both of which propose a standard benchmark and framework allowing dual-EE solutions to be evaluated across a common set of criteria. Authors in [8] construct their comparative framework on security functions which they define to cover the security risks for enterprise mobile applications. On the other hand, we construct our comparative framework on MILS, a well-known trusted model. The main advantage of using MILS is to provide a deeper comprehension of many hidden properties of the dual-EE approach. In addition, some may argue the impartiality of any framework built on self-cooked criteria, while the relevance and the objectivity of our criteria are guaranteed, since they are based on a thoroughly defined trusted model.

8 Conclusions and Future Work

In this paper, we revisited the dual-EE approach, a model that allows mobile smart devices to guarantee a tamper-resistant execution for highly sensitive applications. We introduced the dual-EE approach in the context of trusted computing. The domain of trusted computing gives us convenient abstract models to better represent the characteristics of the dual-EE approach.

In this paper, we also provided a general classification of the dual-EE solutions defined in the literature. The goal of this classification is not to provide an extensive survey, but to examine our framework by applying it on a representative of each class. Results are consistent with related work and sometimes unexpected. They show that TrustZone provides a balanced compromise to implement the dual-EE approach. They also show that systems requiring the maximum level of security should adopt external hardware modules, while hypervisors are ill-adapted to provide high assurance security even though they might improve the overall security level of the system.

We believe that our work can be easily extended to include other comparison criteria. An interesting aspect is the scheduling techniques present on MILS. In some smart devices, it is necessary that malicious allocation of hardware resources (e.g. CPU time) do not impact the SEE execution. Despite their high importance, temporal constraints are rarely taken into account in dual-EE solutions. Our abstract model forms a theoretical basis that systematizes the design of dual-EE solutions regarding primitives defined in the MILS architecture.

Some might think that dual-EE is nothing but a special case of the multi-EE approach in which an arbitrary number of execution environments runs on the SK layer. However, we prove by induction that the opposite is true: all dual-EE solution can construct a multi-EE architecture. Due to space constraint, we do not include our proof in this paper. Future work will focus on extending our model to include more properties related to the SK layer, a comprehensive evaluation of more dual-EE solutions and formal proofs related to our work.

References

1. ARM Holdings plc. Annual report 2013: Strategic report, 2013.
2. Trustonic. `https://www.trustonic.com`, 2014. Accessed: January 2 2015.
3. J. Alves-Foss, P. W. Oman, C. Taylor, and W. S. Harrison. The MILS Architecture for High-Assurance Embedded Systems. *International Journal of Embedded Systems*, 2(3):239–247, 2006.
4. ARMLtd. ARM Security Technology - Building a Secure System using TrustZone Technology, 2009.
5. N. Asokan, J. E. Ekberg, K. Kostiainen, A. Rajan, C. Rozas, A. R. Sadeghi, S. Schulz, and C. Wachsmann. Mobile Trusted Computing. *Proceedings of the IEEE*, 102(8):1189–1206, Aug 2014.
6. A. M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen. Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 90–102, New York, NY, USA, 2014. ACM.
7. C. Dall and J. Nieh. KVM/ARM: The Design and Implementation of the Linux ARM Hypervisor. *SIGPLAN Not.*, 49(4):333–348, Feb. 2014.
8. M. El-Serngawy and C. Talhi. Securing Business Data on Android Smartphones. In *Mobile Web Information Systems*, volume 8640 of *Lecture Notes in Computer Science*, pages 218–232. Springer International Publishing, 2014.
9. W. Enck, M. Ongtang, and P. McDaniel. Understanding Android Security. *IEEE Security and Privacy*, 7(1):50–57, Jan. 2009.
10. T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A Virtual Machine-based Platform for Trusted Computing. *SIGOPS Oper. Syst. Rev.*, 37(5):193–206, Oct. 2003.
11. M. Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold Co., New York, NY, USA, 1988.
12. D. Grawrock. *The Intel Safer Computing Initiative: Building Blocks for Trusted Computing*. Books by engineers, for engineers. Intel Press, 2006.

13. K. Gudeth, M. Pirretti, K. Hoeper, and R. Buskey. Delivering Secure Applications on Commercial Mobile Devices: The Case for Bare Metal Hypervisors. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 33–38, New York, NY, USA, 2011. ACM.
14. J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest We Remember: Cold-boot Attacks on Encryption Keys. *Commun. ACM*, 52(5):91–98, May 2009.
15. J.-Y. Hwang, S.-b. Suh, S.-K. Heo, C.-J. Park, J.-M. Ryu, S.-Y. Park, and C.-R. Kim. Xen on ARM: System Virtualization Using Xen Hypervisor for ARM-Based Secure Mobile Phones. In *Proceedings of the 5th IEEE International Conference on Consumer Communications and Networking*, CCNC '08, pages 257–261, Jan 2008.
16. G. Klein, J. Andronick, K. Elphinstone, T. Murray, T. Sewell, R. Kolanski, and G. Heiser. Comprehensive Formal Verification of an OS Microkernel. *ACM Trans. Comput. Syst.*, 32(1):2:1–2:70, Feb. 2014.
17. K. Kostiainen, J.-E. Ekberg, N. Asokan, and A. Rantala. On-board Credentials with Open Provisioning. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 104–115, New York, NY, USA, 2009. ACM.
18. B. W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8(1):18–24, Jan. 1974.
19. D. Lie, C. A. Thekkath, and M. Horowitz. Implementing an Untrusted Operating System on Trusted Hardware. *SIGOPS Oper. Syst. Rev.*, 37(5):178–192, Oct. 2003.
20. V. R. Pandya and M. Stamp. iPhone Security Analysis. *Journal of Information Security*, 1(2):74–87, Oct. 2010.
21. J. M. Rushby. Design and Verification of Secure Systems. *SIGOPS Oper. Syst. Rev.*, 15(5):12–21, Dec. 1981.
22. N. Santos, H. Raj, S. Saroiu, and A. Wolman. Using ARM Trustzone to Build a Trusted Language Runtime for Mobile Applications. *SIGARCH Comput. Archit. News*, 42(1):67–80, Feb. 2014.
23. A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: Verifying Code Integrity and Enforcing Untampered Code Execution on Legacy Systems. *SIGOPS Oper. Syst. Rev.*, 39(5):1–16, Oct. 2005.
24. G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. AEGIS: Architecture for Tamper-evident and Tamper-resistant Processing. In *Proceedings of the 17th Annual International Conference on Supercomputing*, ICS '03, pages 160–171, New York, NY, USA, 2003. ACM.
25. M. W. Vanfleet, J. A. Luke, W. R. Beckwith, C. Taylor, B. Calloni, and G. Uchenick. MILS: Architecture for High-Assurance Embedded Computing. *CrossTalk: Journal of Defence Software Engineering*, 18(8):12–16, 2005.
26. A. Vasudevan, E. Owusu, Z. Zhou, J. Newsome, and J. M. McCune. Trustworthy Execution on Mobile Devices: What Security Properties Can My Mobile Platform Give Me? In *Proceedings of the 5th International Conference on Trust and Trustworthy Computing*, TRUST'12, pages 159–178, Berlin, Heidelberg, 2012. Springer-Verlag.
27. P. Wilson, A. Frey, T. Mihm, D. Kershaw, and T. Alves. Implementing Embedded Security on Dual-Virtual-CPU Systems. *IEEE Des. Test*, 24(6):582–591, Nov. 2007.