TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 01/10 pp447-457
Volume 22, Number 5, October 2017

# A TrustEnclave-Based Architecture for Ensuring Run-Time Security in Embedded Terminals

Rui Chang*, Liehui Jiang, Wenzhi Chen, Yaobin Xie, and Zhongyong Lu

**Abstract:** The run-time security guarantee is a hotspot in current cyberspace security research, especially on embedded terminals, such as smart hardware as well as wearable and mobile devices. Typically, these devices use universal hardware and software to connect with public networks via the Internet, and are probably open to security threats from Trojan viruses and other malware. As a result, the security of sensitive personal data is threatened and economic interests in the industry are compromised. To address the run-time security problems efficiently, first, a TrustEnclave-based secure architecture is proposed, and the trusted execution environment is constructed by hardware isolation technology. Then the prototype system is implemented on real TrustZone-enabled hardware devices. Finally, both analytical and experimental evaluations are provided. The experimental results demonstrate the effectiveness and feasibility of the proposed security scheme.

**Key words:** run-time security; trusted execution environment; hardware isolation; TrustZone

## 1 Introduction

Embedded terminals (e.g., smart hardware, wearable devices, and mobile devices) have recently attracted great attention from members of the cyberspace security community. On the one hand, embedded systems are already considered the central part of control and weapons systems in the military field. On the other hand, sundry embedded devices have been used by several infrastructure control facilities in the civil domain as well as in automobile control, industrial control, transportation system, electric system, financial system, mobile communication, and so forth.

With the rapid development of Internet of Things (IoT) technology and improvements in the computation performance of mobile embedded devices, the new pattern of informatization application has emerged (e.g., Industrial 4.0, BYOD (Bring Your Own Device), etc.). Embedded terminals, such as smart devices in enterprise office network, smart hardware, as well as wearable and mobile devices, have also received great attention from researchers and practitioners.

Typically, such devices use universal hardware and software to connect with public networks via the Internet, and are probably open to security threats from Trojan viruses and other malware. As a result, the security of sensitive personal data is threatened, and economic interests in the industry are compromised. How do we handle the security problems of complex embedded devices without killing innovation?

Different from traditional personal computers, embedded architecture is limited by its functions and resources. Mature security protection theories and technologies cannot be directly applied to the protection mechanisms of embedded devices. In fact, the security problems of embedded systems are much more complicated than those of desktop systems. Finding ways to improve the security of embedded terminals is an urgent and challenging problem for researchers.

• Rui Chang, Liehui Jiang, and Yaobin Xie are with the State Key Laboratory of Mathematic Engineering and Advanced Computing, Zhengzhou 450001, China. E-mail: crix1021@163.com.
• Wenzhi Chen and Zhongyong Lu are with the Department of Computer, Zhejiang University, Hangzhou 310027, China. E-mail: chenwz@zju.edu.cn.
∗ To whom correspondence should be addressed.
Manuscript received: 2016-09-28; accepted: 2016-10-20

Security vulnerabilities from an Operating System (OS) or third-party software are becoming increasingly serious. Protecting run-time systems by preventing vulnerabilities and patching faulty programs is no longer effective. Unfortunately, there is a lack of mature theories and fundamental studies that focus on the run-time security of embedded devices. The mainstream implementation schemes are based on virtualization technology and secure coprocessors. Virtualization technology utilizes a supervisor to manage system resources to achieve Virtual Machine (VM) introspection. The supervisor monitors the conditions of Guest Virtual Machine (GVM) in real time and detects potential kernel attacks. However, the VM supervisor has more vulnerabilities than the OS due to its complexity. Furthermore, the performance overhead resulting from hardware virtualization is unacceptable for the computation capability of embedded devices. Additionally, not all the embedded devices have virtualization support. A scheme for detecting kernel vulnerabilities based on a secure coprocessor has already been proposed[1], but it only supplies an isolated execution environment with a lack of controlling capability on system resources, such as memory and other exterior equipment. This approach can lead to two negative results. One is that monitoring function deployed in a kernel's address space can be easily tampered by attackers. Consequently, real-time monitoring would fail. The other is that the scheme based on a secure coprocessor can only detect system status but cannot manage and control abnormal behaviors. For example, the integrity measurement scheme of Linux from IBM can measure and verify the running processes[2], but it cannot prevent the execution of illegal processes.

To effectively address the security problems for embedded terminals, this paper explores several key technologies of OS support for run-time security (Section 3). It proposes a TrustEnclave-based secure architecture on embedded terminals, and presents the implementation scheme (Section 4). The major advantage of the proposed architecture is that it builds a TrustEnclave in the address space of the OS kernel, which cannot be tampered by the untrusted OS kernel itself. TrustEnclave, which is protected by hardware isolation technology, is an area of OS kernel. We implement the prototype system on real TrustZone-enabled hardware devices, construct a trusted execution environment by hardware isolation technology, and provide both analytical and experimental evaluations (Section 5). The experimental results demonstrate that the proposed security scheme is effective and feasible. Hence, we can expect that the proposed architecture and implementation scheme would better support potential applications on embedded terminals where run-time security is desired (e.g., smart devices).

The main contributions of this paper are as follows:

(1) We explore mainstream technologies in recent years and compare existing implemented schemes;

(2) We propose a novel TrustEnclave-based secure architecture on embedded terminals, which builds a TrustEnclave in the address space of the OS kernel that cannot be tampered by the untrusted OS kernel; and

(3) We implement the prototype system on real TrustZone-enabled hardware devices, and present both analytical and experimental evaluations.

## 2 Overview

An embedded system is a custom-built measurement system with demanding functions, reliability, cost, volume, and power dissipation. It consists of an embedded microprocessor, hardware platform, embedded OS, and various other applications. An embedded system is similar to a computer system and features three security attributes of confidentiality, integrity, and availability. The explanations of these attributes differ greatly based on their environment.

What is a Trusted Execution Environment? Before we answer this question, we need to first define execution environments in general. At a high level of abstraction, an execution environment is the software layer running on top of a hardware layer. Both hardware and software layers are combined to form a device. We focus on a class of devices that contain two execution environments, which are physically separated. One environment contains the main OS and applications, and the other environment contains the trusted software components. Thus, we have a physical separation between the Trusted Area and the Untrusted Area. The trusted area is not intrinsically trusted; no untrusted software executes in it, and no hardware is attached to it. Thus, it offers a stronger guarantee than an equivalent outside of the security perimeter. However, given that the trusted area is separated by the hardware from OS and applications, its isolation is guaranteed, that is, everything outside the trusted area is untrusted. Each area features a different execution environment. In other words, a device has two different software stacks. We denote the execution environment in the trusted area Trusted Execution Environment (TEE), and the one in the untrusted area Rich Execution Environment (REE). The indeterministic software in the REE cannot affect the software running in the TEE.

Run-time security supplies an isolated secure execution environment (i.e., TEE), where the confidentiality and integrity of the code and data are guaranteed. The secure characteristics include isolated execution, execution files integrity, run-time code integrity, control flow integrity, and so on. The protected resources in run-time security are OS kernel, memory, user process, files, peripherals, etc.

## 3 Background and Motivation

The research on key technologies of OS support for run-time security focuses on virtualization technology, Trusted Platform Module (TPM), Intel Software Guard Extensions (SGX), and ARM TrustZone.

Owing to hypervisors with higher privileges compared with OS, the security enhancement scheme based on virtualization technology enhances system security by isolating and monitoring. Such a scheme usually deploys a monitoring tool outside the system; thus, such a tool cannot be manipulated by malicious software. Furthermore, the untrusted software running in special virtual machine, which are likely to be manipulated by malicious software, cannot bypass hypervisors and influence other virtual machines. Secvisor utilized the Secure Virture Machine (SVM) of AMD processor to supply run-time kernel code integrity protection, resulting in greater performance overhead, and is not portable for embedded systems[3]. At present, the virtualization products of the mobile embedded terminal field are vmware, L4Android, OKL4, Xen, LXC, and so on. The Arc Lab of Zhejiang University utilized LXC in Android 4.0 to implement a lightweight virtual machine scheme[4], which isolated applications with different security levels. Given that several virtual machines still shared a sole kernel, the scheme proved to be insufficient because it was unable to supply the solution for kernel attacks.

Zheng et al. designed and implemented a scheme for a trusted mobile terminal based on hardware platform with an OMAP730 processor. Chen et al.[6] proposed a trusted mobile platform architecture based on a Mobile Trusted Module (MTM) and gave the formal verification based on predicate logic. Zhao et al.[7] from Wuhan University designed and implemented a trusted PDA based on chip JetWay2810. Kim et al. from Korea implemented a highly efficient hardware architecture with SHA-1 and HMAC in 2007; in 2010, they designed the first small sized MTM chip with triple calculating speed of the current TPM and less energy consumption[8, 9]. TPM uses secure key, and anything untrusted did not know the key; thus, anything encrypted by the key was considered secure[10]. However, it cannot defend the system against run-time attacks. SGX (Intel Software Guard Extensions) and TrustZone respectively adopted different methods for run-time security.

Intel SGX is a set of new CPU instructions that can be used by applications to set aside private isolation regions of code and data[11]. It enables applications to preserve the confidentiality and integrity of sensitive code and data without disrupting the ability of the legitimate system software to schedule and manage the use of platform resources. SGX helps to define secure regions of code and data that maintain confidentiality even when an attacker has physical control of the platform and can conduct direct attacks on OS, VMM, and memory. SGX adds 18 instructions to extend Intel ISA (Instruction Set Architecture) to ensure software security. Given that SGX is considered the latest security technology of Intel since 2013, utilizing it on embedded platforms has numerous potential applications and thus deserves greater attention by researchers. Professor Ahmad-Reza Sadeghi from Technische Universit Darmstadt (CASED) of Germany pursued his studies on Trusted Execution Environments of embedded system security and presented the theoretical analysis for embedded system security with Intel SGX support[12]. Researchers from the Georgia Institute of Technology achieved a project, openSGX, which simulated SGX by QEMU, thus marking the first attempt to use SGX in embedded fields[13].

ARM defines TrustZone[14] as a hardware-supported system-wide approach to security that is integrated in high-performance processors, such as Cortex-A9, Cortex-A15, and Cortex-A12[15]. Today, TrustZone is implemented in most modern ARM processor cores including the ARM1176, Cortex-A5/A7/A8/A9/A15, and the newest ARMv8 64-bit Cortex-A53 and Cortex-A57. TrustZone supplies isolated execution environments for key system modules and protects system resources in security working mode. Compared with complex hypervisors, TrustZone is a more appropriate method for ensuring embedded system security.

While TrustZone[16] has been introduced more than 10 years ago, it is only recently that hardware manufacturers (e.g., Xilinx, Nvidia, or Freescale) and software solutions (e.g., Open Virtualization 19, TOPPERS SafeG20, Genode21, Linaro OP-TEE, T622, or Nvidia TLK) have respectively proposed hardware platforms and programming frameworks that enable researchers[17] and industry practitioners to experiment and develop innovative solutions with TrustZone. This development has led to a more open TrustZone technology.

## 4 Design and Implementation

### 4.1 TrustZone-based TEE architecture

In order to support TEE, a device must be able to define a security perimeter separated by hardware from the main OS and other applications, in which only the trusted code executes. We present the TrustZone-based TEE architecture in Fig. 1. We refer to this security perimeter as a trusted area called the Secure World (SW). The trusted area is represented on the right side of the figure (blue), where trusted components execute in TEE. All components outside the trusted area form the untrusted area called the Normal World (NW), where OS and applications execute in REE. The untrusted area is represented on the left side of the figure (yellow). Peripherals connected to the system bus belong to either of the two areas, or both of them, depending on the specific technology. TrustZone relies on the so-called NS bit, an extension of the AMBA3 AXI system bus to separate the execution between SW and NW.

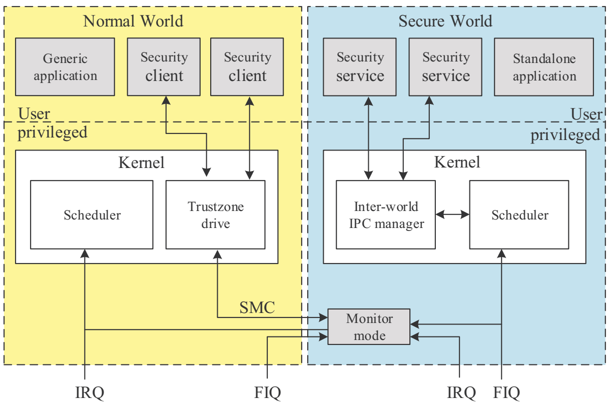


**Fig. 1 TrustZone architecture.**

### 4.2 Design challenges

The most powerful feature of TrustZone is that it is capable of securing any peripherals connected to the system bus (e.g., interrupt controllers, timers, and user I/O devices) in a way that they are only visible from the SW. One of the most difficult points is to gain the code, data, and real-time status from any part of NW. When real-time protection is turned on, it will not only prevent attacks by modifying kernel effectively, but also defend the attacks when the two logical pages from different processes are allocated to the same physical page with malicious kernel behaviors.

In order to deprive NW of gaining access to hardware, the support from hardware includes two aspects. One is that a higher privilege code cannot run in the lower privilege mode. The other is the PXN (Privileged eXecution Never) mode supported by ARM's virtual memory management. By setting the value of the flag bit, we can control the range of physical address space where the privileged code is running. For example, the instructions LDC and MCR, which access memory by register, only run in the special segment of memory space.

Then, we implement three technical points as follows. First, sensitive codes that can modify the crucial state of hardware only run in the security memory space in plan. Second, the security physical memory space cannot be modified. Third, there is no such address where it is possible to obtain a protosomatic sensitive code in the security physical memory space. This means that we must artificially recode the sensitive code. Executing the sensitive code by jumping into security space is impossible, and the result is two-fold: NW cannot execute the sensitive code from normal memory because there is no sensitive code in the normal memory, and the sensitive code cannot execute in NW because NW cannot read sensitive code from SW. Recoding can be achieved by two approaches, similar to binary translation in full-virtualization and kernel modification in para-virtualization. The cost of binary translation for ARM is lower than that for X86 because of ARM's 32 bit fixed instruction format. As the NW ultimately executes the sensitive code, and the

CPU actually executes an SMC call. When NW receives an SMC call, it checks the value of the previously saved register, which is taken as the operation code, and jumps according to the protocol. The hardware functions are actualized in SW. Consequently, the instructions in NW have the same functions as before and are also secured.

## 4.3 TrustEnclave-based privilege mode

TrustZone introduces new states of security for ARM architecture, which decide whether the operations occur in SW or NW. The hardware of SW has special design for strengthening security, while it can isolate codes in hardware conditions. Security software supplies basic security services, meanwhile it provides interface to link any other nodes of security chain, including smart card, OS, and normal applications. In general, the ARM processor has seven work modes divided into two categories, namely, user mode and privileged modes. Access rights to certain resources are restricted in the user mode, but they are not constrained in the other six privileged modes.

- User mode: Low-privileged mode, where the user code outside the system code runs;
- System mode: Privileged code running in system mode;
- Management mode: System using mode;
- DataAbort: Access data error;
- Fast interrupt: Rapid response to external interrupt;
- External interrupt: Normal interrupt mode;
- Undefined: Illegal instructions being executed.

To improve the above mentioned design, the TrustZone-based ARM processor adds security and non-security modes to differentiate the state of processor. It adds a new processor mode (i. e., Monitor mode) to be privileged mode and user mode and then differentiates the state of the processor by the lowest bit of coprocessor C1 (i. e., NS bit). If NS bit=0, it is secure and trusted; if NS bit=1, it is non-secure and untrusted. The register can be accessed if and only if it is privileged and in the security mode. The operational principle of differentiating security and non-security is similar to that of differentiating between the privileged and user modes. The NS bit not only affects CPU core and memory subsystem, but also affects the functions of the peripherals on the chip.

The NS bit indicates the current running state of kernel. The independently running mode (i.e., monitor mode) of the processor is used to control the security state of system, instructions, and access authority. It switches between security and normal states by modifying the NS bit. Moreover, it saves the current context state and clears the registers as needed. The eight new-processor modes with NS bit are shown in Table 1. Each mode of the ARM processor corresponds to an interrupt vector table. The offset addresses of interrupt vector tables are shown in Table 2.

As shown in Table 2, system call (SVC) and security call (SMC) use the same interrupt vector address. SVC is used to switch the user mode to the privileged mode, whereas SMC is used to switch the privileged mode to the security mode. However, undefined instruction exception may ocur if SMC is called in the user mode.

The security feature of TrustZone can be used in sundry safety applications. The extended security features must be satisfied by the fundamental principles below.

(1) Define a new operation switching security and non-security state; the majority of codes run in NW, and only trusted codes run in SW.

(2) Set part of memory space as security space; access

**Table 1 The mode list of TrustZone support.**

| Mode | Privilege level | State | |
|---|---|---|---|
| | | NS bit=1 | NS bit=0 |
| User mode | User | Untrusted | Trusted |
| Fast interrupt | Privileged | Untrusted | Trusted |
| Common interrupt | Privileged | Untrusted | Trusted |
| Privileged mode | Privileged | Untrusted | Trusted |
| Illegal access | Privileged | Untrusted | Trusted |
| Undefined | Privileged | Untrusted | Trusted |
| System | Privileged | Untrusted | Trusted |
| Monitor | Privileged | Trusted | Trusted |

**Table 2 Interrupt vector table.**

| Interrupt exception types | Mode | Offset address |
|---|---|---|
| Reset | Privileged mode | 0x00 |
| Undefined | Undefined mode | 0x04 |
| System call | Privileged mode (SVC) | 0x08 |
| Secure call | Monitor mode (SMC) | 0x08 |
| Prefetch failure | Illegal access | 0x0c |
| Access error | Overflow | 0x10 |
| Common interrupt | Common interrupt | 0x18 |
| Fast interrupt | Fast interrupt | 0x1c |
| Reset | Privileged mode | 0x00 |

SW only in the security state.

(3) Strictly control the entry of entering SW.

(4) Quitting from SW needs to be restricted.

We can modify the NS bit only in the privileged mode, that is, we can switch the state from the security mode to the non-security mode by setting the NS bit. On the contrary, we cannot switch from non-security mode to security mode, because the NS bit cannot be modified in the non-security mode.

If it is in non-security mode, calling system call SMC is the only way to enter the security mode. Yet if it is both in the user and non-security mode, it must call the SVC first. It is worthy to note that the modification of modes is severely restricted. If it is in the non-security mode, calling security call SMC is the only way to change into the monitor mode; meanwhile, when it is both in the privileged and security modes, we can modify the system mode directly. All hardware resources can be accessed in the monitor mode.

## 4.4 TrustEnclave construction

We construct the protected isolation TrustEnclave, and make corresponding authority policies of page table mapping. The structure of TrustEnclave is shown in Fig. 2. Here, SW is represented on the right side of the figure (blue), where trusted components execute in TEE, and NW is represented on the left side of the figure (yellow). TrustEnclave is the enclave that is in the NW's address space with normal privilege level, but is protected by the trusted isolation environment. The monitor codes and TrustEnclave could not be tampered by the attacker.

The greatest challenge here is to protect the monitor code in the NW's address space. Given that NW has full control over its own system resources (e.g., physical memory, page table, and corresponding control register), it is possible to bypass the security monitor. The SW must track the behavior of the monitor codes and construct TrustEnclave in the kernel space. The two specific procedures are disposition of monitor points and isolation protection of the monitored area. It will create a security world process control block (i. e., sw_pcb) as each valid process is created inside SW. The sw_pcb manages the state information of process, including process page table base address, physical address of the security shared memory, process shadow stack, process jump record table, etc. The sw_pcb can be used to provide help for the proof procedure of security policy. Binary codes are recoded during kernel image loading; meanwhile, system image files need not be modified. The instruction set of ARM architecture has a fixed-length (e.g., 16-bit in Thumb and 32-bit in ARM), and the instruction addresses are one-byte aligned. SW can pre-acquire the kernel address space arrangement, making it easier to locate and identify the location of the correlative code, and providing a facility to recode binary codes. The monitored kernel codes are replaced by SW with SMC instructions, and the monitor point type is identified by the 4-bit immediate operand of SMC instructions. Referencing the management mechanism of shadow page table in virtual technology, all physical memory mappings that include the page table are compulsively read-only. Whenever a kernel updates the page table, this will trigger data abort exceptions due to page permission errors and jump to the exception vector table executing the exception handler. Thus, we insert a monitor point into the data abort exception of the exception vector table. Doing so ensures that all updates of page tables are intercepted by SW.

In order to ensure that the memory page table is mapped read-only, we add a new security strategy while switching Translation Table Base Register (TTBR) and updating page table, that is, we must ensure that all physical memory page tables are read-only and that the writable multimap does not exist. This requires recording the physical address when all the page tables are created in SW. ARM-Android uses the two-level page table by default. In the following two situations, the first-level page table is created. One situation is initializing the page table of a kernel (i.e., swap_pg_dir) itself and trying to write it into TTBR. The other is the first time a process is scheduled to execute after creation when TTBR is switched. The second-level

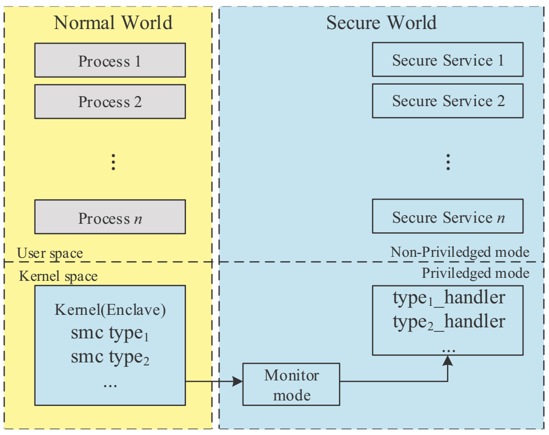


**Fig. 2 Structure of TrustEnclave.**

physical page table is created when the first level page table is updated. Both can be intercepted by the existing monitor points. Thus, the security strategies above can be validated effectively by SW. We should insert two kinds of monitor points into NW: control register modification (MMU, WXN, TTBR) and data abort exception.

## 5 Evaluation

### 5.1 Analytical evaluation

We provide an analytical evaluation of our contributions. We first provide an exhaustive security analysis for each of them, after which we look at the design requirements we established and study how they are met in SW. From a software point of view, any design of a trusted service using TrustZone should rely on the following three main components: (1) the trusted operating system which represents a specific way to organize the TrustZone's secure world, and a commodity OS that supports the execution of complex untrusted applications (i.e., innovative services); (2) a TrustZone driver that enables interactions between secure and non-secure worlds; and (3) a set of trusted modules that implement the trusted services in the TrustZone secure world.

Our design advocates for a high integration between the two areas to support innovative services, inevitably exposing the components in the trusted area. Still, we maintain the assumption that the untrusted area (i.e., untrusted applications and commodity OS) is completely untrusted, and the fact that it is compromised does not affect neither the confidentiality nor the integrity of the sensitive assets.

The generic TrustZone driver and the secure monitor are two TrustZone components that are compromised as soon as they are exposed to the untrusted area. These two components are closely related, because their locations in the untrusted area respond to two different attack vectors. A third attack vector that we cover is directly compromising the secure area without using the interfaces exposed to the untrusted area.

In this analytical evaluation, we show our contributions, i.e., resist a large percentage of the attack vectors that we know of today, and comply with the requirements we have established for them in our design. Indeed, we have satisfied our main objective: increasing the security of an embedded system with possible theoretical complex applications without killing innovation. The untrusted area, where innovative applications are executed, can be fully compromised. However, through a series of run-time security primitives, these applications can access trusted services while guaranteeing the confidentiality and integrity of sensitive data. More importantly, these trusted services not only enable the outsourcing of secure tasks to a trusted area protected by hardware, they also allow sensitive data to leave such trusted area and access to innovative, untrusted services, while still guaranteeing its confidentiality and integrity.

### 5.2 Experimental evaluation

As mentioned above, when we started experimenting with TrustZone, options are limited by both hardware and software. We rely on the CES-4412P development board, which is formed around Samsung's newest Exynos4412, that is, a quad-core ARM Cortex-A9 processor. The experimental platform is one of the few platforms that can fully support TrustZone and where TrustZone registers are available. More concretely, we use the CES-4412P development board, which runs typically at 1.4–1.6 GHz with 32 KB L1 cache and 1 MB L2 cache. The CES-4412P is depicted in Fig. 3.

In our experiments, the TrustZone operating system is Sierraware's GPL version of Open Virtualization. We use Linux kernel (version 4.0.1) as the operating system running in the NW, together with a light command-based version of Ubuntu. These systems respectively manage the secure space, kernel space, and user space.

Every time a secure task (or a trusted module) is called from the kernel space, a context switch takes place between the kernel and secure space. Even though this process is implementation specific, it at least

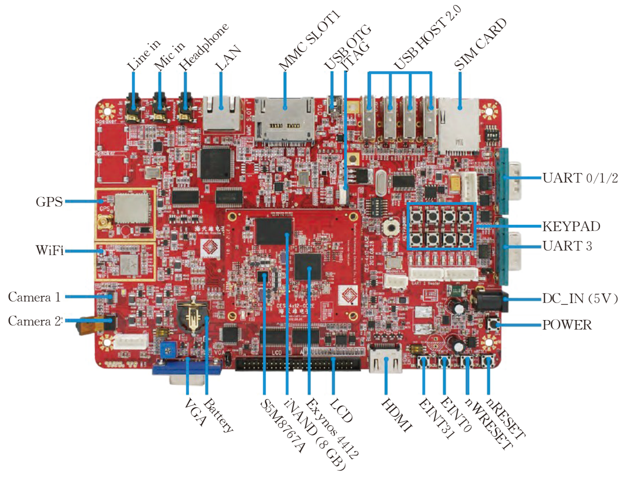


**Fig. 3 Samsung CES-4412P development board.**

involves the following: saving the untrusted state, switching software stack, loading the secure state, dispatching the secure task, and returning to the kernel space (save secure state, change software stack, load untrusted sate). We denote this double context switch as Secure Round Trip. The metric we use, which is defined below, is the overhead introduced by the secure space.

$$\text{Overhead} = \frac{T_{\text{secure}} - T_{\text{kernel}}}{T_{\text{kernel}}} \quad (1)$$

We present a comprehensive evaluation for the influence of our work by Lmbench, i.e., embedded platform evaluation tool. It evaluates the switching privileged mode, memory mapping, page fault exception handling, and so on. We compare the execution efficiencies of system call between the original OS and the TrustEnclave-based one. Then, we calculate the overhead. The experimental results are shown in Fig. 4.

## 6 Discussion and Future Work

Different from the schemes based on secure coprocessor, TEE architectures provide a secure processor environment, wherein a single core supports multiple virtual cores that are mutually exclusive of one another, i.e., when one is running, the other is suspended. Generally, there is a kind of trigger that allows the core to switch from one state to the other. We implement one of the TEE architectures, which is different from those presented by other international studies. The comparison results are shown in Table 3.

Based on our experience in designing and building support for trusted embedded terminals, we now propose a roadmap for future works. As demonstrated in the current work, hardware isolation is indeed an effective solution for providing run-time security in commodity OS without making assumptions on their trustworthiness. Meanwhile, it also introduces an affordable overhead in terms of performance. On top of our initial hypothesis, our future work includes utilizing sensitive assets, and serving as a basis for usage policy enforcement via hardware isolation.

We divide this roadmap for future work in three sections. First, we would like to further improve the current architecture in terms of OS support and security modules. Second, it would be interesting to make improvements upon the current protection modules. Here, we take the threat model and memory protection mechanism separately. Finally, we would also like to provide the memory integrity verification by formalization in the future.

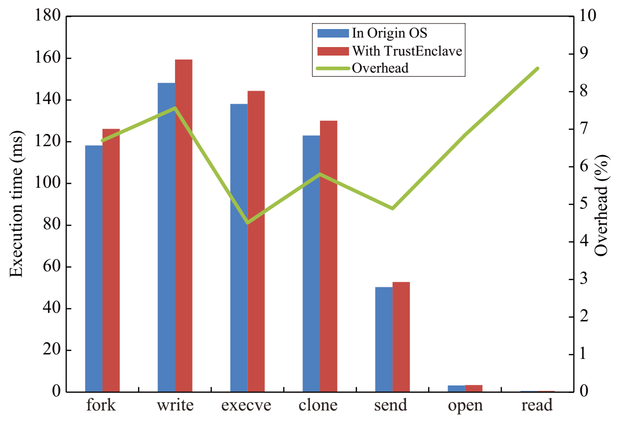


**Fig. 4 Performance evaluation for TrustEnclave.**

## 7 Related Work

In terms of size and overhead, the separate TPM chip in embedded terminals is inadequate. TPM module implemented in software is another choice. Aaraj et al.[23] tested the overhead and execution time of software TPM instructions on PDA. Choi et al.[24] from Korea implemented a mobile trusted system based on micro-kernel. Bugiel and Ekberg[25] from Sweden introduced the Dynamic Root of Trust for Measurement (DRTM) to protect and measure MTM, which attempted to establish a dynamic trusted computing environment. Meanwhile, Ekberg et al.[26] from the Nokia Research Institute implemented the

**Table 3 Comparison of TEE architectures with international researchers.**

| Researchers | Secure World | Normal World | Platform | Hardware-assistent | Memory protection |
|---|---|---|---|---|---|
| Gonzlez and Bonnet[17] | Open Virtualization | Linux3.8.0 | Xilinx ZC702 | Yes | Implement |
| Pinto et al.[18] | FreeRTOS | Linux | Xilinx ZC702 | Yes | Not-mentioned |
| McGilliion et al.[19] | Linux | Android/IOS/Linux | Open-TEE | No | Not-mentioned |
| Winter et al.[20] | Linux | Android/Linux | QEMU emulator | No | Not-mentioned |
| Yang et al.[21] | T-OS(Trust-E) | Android4.0 | SMDK210 | Yes | Not-mentioned |
| Zhang et al.[22] | Open Virtualization | Linux2.6.35 | Xilinx 7000 | Yes | Mentioned |
| Our work | Open Virtualization | Linux4.0.1 | CES-4412P | Yes | Implement |

simulator MTM based on simulator TPM. MIPS developed security processor core, including extended ISA, to accelerate encryption and decryption functions and security memory management. IBM produced a coprocessor distorting authentication. Zhang et al.[27] from Georgia Mason University studied a probable run-time attack and proposed the implantation scheme to solve it. Smolyar et al.[28] from Technion aimed at SRIOV utilizing a VM to control another VM.

In recent years, academic researchers focused on an ARM-Android platform for embedded terminals security[29], and new technologies and ideas emerged within the combination of academia and industry[30]. TrustZone-based technologies have also been applied to the mobile terminal field[31], including Apple SecureEnclave, Samsung KNOX, and so forth[32]. After establishing Hypervisor-Based IMA (i.e., HIMA), Azab et al.[33] from North Carolina State University developed the applications for KNOX and explored new related technologies. Ge et al.[34] proposed a protection scheme for kernel integrity on mobile embedded devices based on TrustZone without implementation. On a .Net platform, a security scheme based on TrustZone and TEE was jointly developed by Microsoft Research and Lisbon University[25]. Researchers from CASED proposed a new code provisioning paradigm for the codes that are intended to run within execution environments, and is established on top of secure hardware[35].

In addition, Ruhr-Universitaet and Microsoft Research Bochum utilized the newest SGX secure mode[36] to isolate the physical memory of individual nodes and implemented trustworthy data analytics in the cloud[37]. Jin et al.[38] from the Korea Advanced Institute of Science and Technology proposed a scheme to monitor hypervisor and protect client resources with hardware assistance.

## 8 Conclusion

At the beginning of this paper, we argue that one of the main factors enabling cyberattacks includes the increasing complexity of OS and software. Our assertion is that complexity hides vulnerabilities in the code, causing the software to occasionally behave in a non-deterministic manner. In our view, cyberattacks are indeed about detecting unspecified behaviors and finding ways to exploit them. The question that we asked, and one which motivated our work, was: How do we handle the security problems of complex embedded devices without killing innovation?

We tried to answer this question by focusing on run-time security. With more system vulnerabilities and much complex network environment, trusted kernel hardly exists in execution. The key technologies of OS support for run-time security have become research hotspots. An efficient and feasible implementation scheme is presented in this work. Specifically, we proposed an architecture with which to construct a trusted execution environment isolated from OS kernel by hardware isolation technology for embedded terminals. The major advantage of the proposed architecture is that it builds a TrustEnclave in the address space of the OS kernel, which cannot be tampered by the untrusted OS kernel itself. TrustEnclave, which is protected by hardware isolation technology, is an area of OS kernel. Hence, a system monitor program should be trusted. Our experiments demonstrate that the proposed security scheme is effective and feasible. The security scheme can be used to protect memory, prevent malicious applications, insulate sensitive data, and deal with some other problems in the field of embedded system security. We can expect that the proposed architecture and implementation scheme can provide better support for potential applications on embedded terminals where run-time security is desired (e.g., smart devices).

**Acknowledgment**

Many thanks to Xian Chen and Yuxia Cheng for their helpful discussion about this work. This work was supported by the National Natural Science Foundation of China (Nos. 61572516 and 61503213).

## References

[1] R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn, Design and implementation of a TCG-based integrity measurement architecture, in *Usenix Security Symposium*, San Diego, CA, USA, 2004, p. 16.

[2] A. M. Azab, P. Ning, E. C. Sezer, and X. Zhang, HIMA: A hypervisor-based integrity measurement agent, in *Computer Security Applications Conference*, IEEE Computer Society, 2009, pp. 461–470.

[3] A. Seshadri, M. Luk, N. Qu, and A. Perrig, SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes, *ACM SIGOPS Operating Systems Review,* vol. 41, no. 6, pp. 335–350, 2007.

[4] L. Xu, W. Chen, and Z. Wang, Research about virtualization of ARM–based mobile smart devices,

*Lecture Notes in Electrical Engineering*, vol. 308, pp. 259–266, 2014.

[5] Y. Zheng, D. He, and M. He, Trusted computing based user authentication for mobile equipment, (in Chinese), *Chinese Journal of Computer*, vol. 29, no. 8, pp. 1255–1264, 2006.

[6] S. Y. Chen, Y. Y. Wen, and H. Zhao, Conceptual design of trusted mobile platform, (in Chinese), *Journal of Northeastern University*, vol. 29, no. 8, pp. 1096–1099, 2008.

[7] B. Zhao, H. G. Zhang, J. Li, L. Chen, and S. Wen, The system architecture and security structure of trusted PDA, (in Chinese), *Chinese Journal of Computers*, vol. 33, no. 1, pp. 82–92, 2010.

[8] M. Kim, H. Ju, Y. Kim, J. Park, and Y. Park, Design and implementation of mobile trusted module for trusted mobile computing, *IEEE Transactions on Consumer Electronics*, vol. 56, no. 1, pp. 134–140, 2010.

[9] M. Kim, D. Lee, and J. Ryou, Compact and unified hardware architecture for SHA-1 and SHA-256 of trusted mobile computing, *Personal and Ubiquitous Computing*, vol. 17, no. 5, pp. 921–932, 2013.

[10] D. Z. Shen, X. B. Hu, H. Z. Liu, F. L. Li, F. Liu, Y. Tao, and Z. L. Ye, Security research of state cryptographic authentication security chip in smart grid, in *China International Conference on Electricity Distribution*, Shenzhen, China, 2014, pp. 416–418.

[11] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, Using innovative instructions to create trustworthy software solutions, in *International Workshop on Hardware and Architectural Support for Security and Privacy*, New York, NY, USA, 2013, pp. 1–10.

[12] A. Sadeghi, Trusted execution environments Intel SGX, Available: http://sigops.org/sosp/sosp13/, Accessed on Nov. 18, 2014.

[13] P. Jain and S. Desai, Intel SGX emulation using QEMU, Available: https://github.com/sslab-gatech/opensgx, Accessed on May 15, 2015.

[14] Y. M. Zhou, The analysis of TrustZone secure technology based on ARM architecture, (in Chinese), *Microcomputer Information*, vol. 24, no. 36, pp. 69–71, 2008.

[15] A. Baumann, M. Peinado, and G. Hunt, Shielding applications from an untrusted cloud with haven, *ACM Transactions on Computer Systems*, vol. 33, no. 3, pp. 1–26, 2015.

[16] T. Alves, TrustZone: Integrated hardware and software security, *ARM White Paper*, vol. 3, no. 4, pp. 18–24, 2004.

[17] J. Gonzlez and P. Bonnet, Towards an open framework leveraging a trusted execution environment, in *the 5th International Symposium on Cyberspace Safety and Security*, 2013, pp. 458–467.

[18] S. Pinto, D. Oliveira, J. Pereira, N. Cardoso, M. Ekpanyapong, J. Cabral, and A. Tavares, Towards a lightweight embedded virtualization architecture exploiting ARM TrustZone, in *IEEE International Conference on Emerging Technologies and Factory Automation*, Barcelona, Spain, 2014, pp. 1–4.

[19] B. McGillion, T. Dettenborn, T. Nyman, and N. Asokan, Open-TEE — An open virtual trusted execution environment, in *TRUSTCOM'15 Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA*, Washington, DC, USA, 2015, pp. 58–67.

[20] J. Winter, P. Wiegele, M. Pirker, and R. Tögl, A flexible software development and emulation framework for ARM TrustZone, in *International Conference on Trusted Systems*, Beijing, China, 2011, pp. 1–15.

[21] X. Yang, P. Shi, B. Tian, B. Zeng, and W. Xiao, Trust-E: A trusted embedded operating system based on the ARM Trustzone, in *IEEE 11th Intl. Conf. on Ubiquitous Intelligence and Computing and 11th Intl. Conf. on Autonomic and Trusted Computing and 14th Intl. Conf. on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, IEEE Computer Society, 2014, pp. 495–501.

[22] Y. Zhang, D. Feng, Y. Qin, and B. Yang, A Trustzone-based trusted code execution with strong security requirements, (in Chinese), *Journal of Computer Research and Development*, vol. 52, no. 10, pp. 2224–2238, 2015.

[23] N. Aaraj, A. Raghunathan, and N. K. Jha, Analysis and design of a hardware/software trusted platform module for embedded systems, *ACM Transactions on Embedded Computing Systems*, vol. 8, no. 1, pp. 3296–3306, 2008.

[24] S. Choi, J. Han, J. Lee, J. Kim, and S. Jun, Implementation of a TCG-based trusted computing in mobile device, in *International Conference on Trust, Privacy and Security in Digital Business*, Springer-Verlag, 2008, pp. 18–27.

[25] S. Bugiel and J. E. Ekberg, Implementing an application-specific credential platform using late-launched mobile trusted module, in *STC'10 Proceedings of the Fifth ACM Workshop on Scalable Trusted Computing*, Chicago, IL, USA, 2010.

[26] J. E. Ekberg, N. Asokan, and K. Kostiainen, Method and apparatus to reset platform configuration register in mobile trusted module, European Patent EP2537115, May 13, 2015.

[27] F. Zhang, K. Leach, A. Stavrou, H. Wang, and K. Sun, Using hardware features for increased debugging transparency, in *IEEE Symposium on Security and Privacy*, IEEE, 2015, pp. 55–69.

[28] I. Smolyar, M. Ben-Yehuda, and D. Tsafrir, Securing selfvirtualizing ethernet devices, in *USENIX Conference on Security Symposium*, Washington, DC, USA, 2015, pp. 335–350.

[29] S. Fahl, M. Harbach, T. Muders, L. Baumg, B. Freisleben, and M. Smith, Why eve and mallory love android: An analysis of android SSL (in) security, in *CCS'12: Proceedings of the 2012 ACM Conference on Computer and Communications Security*, Raleigh, NC, USA, 2012, pp. 50–61.

[30] S. H. Kim, D. Han, and D. H. Lee, Predictability of Android OpenSSL's pseudo random number generator, in *ACM Sigsac Conference on Computer and Communications Security*, Berlin, Germany, 2013, pp. 659–668.

[31] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, An empirical study of cryptographic misuse in Android

applications, in *ACM Sigsac Conference on Computer and Communications Security*, Berlin, Germany, 2013, pp. 73–84.

[32] N. Santos, H. Raj, S. Saroiu, and A. Wolman, Using ARM trustzone to build a trusted language runtime for mobile applications, in *International Conference on Architectural Support for Programming Languages and Operating Systems*, Salt Lake City, UT, USA, 2014, pp. 67–80.

[33] A. M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganeshand, J. Ma, and W. Shen, Hypervision across worlds: Real-time kernel protection from the ARM TrustZone secure world, in *ACM Sigsac Conference on Computer and Communications Security*, Scottsdale, AZ, USA, 2014, pp. 1028–1031.

[34] X. Ge, H. Vijayakumar, and T. Jaeger, Sprobes: Enforcing kernel code integrity on the TrustZone architecture, arXiv: 1410.7747, 2014.

[35] A. Dmitrienko, S. Heuser, T. D. Nguyen, M. D. S. Ramos, A. Rein, and A. Sadeghi, Market-driven code provisioning to mobile secure hardware, in *Financial Cryptography and Data Security*, Springer Berlin Heidelberg, 2015, pp. 387–404.

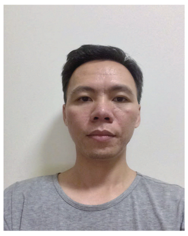
[36] A. Baumann, M. Peinado, and G. Hunt, Shielding applications from an untrusted cloud with Haven, *ACM Transactions on Computer Systems*, vol. 33, no. 3, pp. 1–26, 2015.

[37] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar, and M. Russinovich, VC3: Trustworthy data analytics in the cloud using SGX, in *IEEE Symposium on Security and Privacy*, Washington, DC, USA, 2015, pp. 38–54.

[38] S. Jin, J. Ahn, J. Seol, S. Cha, J. Huh, and S. Maeng, HSVM: Hardware-assisted secure virtual machines under a vulnerable hypervisor, *IEEE Transactions on Computers*, vol. 64, no. 10, pp. 2833–2846, 2015.

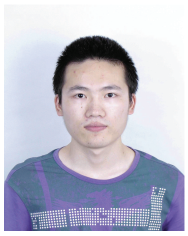
**Rui Chang** received the MS degree from Wuhan University of Technology in 2007, and BA degree from Zhengzhou University in 2003. She is an associate professor with the State Key Laboratory of Mathematic Engineering and Advanced Computing, Zhengzhou, China. She is currently a PhD candidate and visiting scholar at the College of Computer Science and Technology, Zhejiang University. Her research interests include computer architecture, embedded system security, and access control.

**Liehui Jiang** received MS degree in computer science and technology from PLA University of Science and Technology in 1994, the PhD degree and BA degree from the PLA Information Engineering University, China in 1989 and 2007, respectively. He is currently a professor and a PhD supervisor with the State Key Laboratory of Mathematic Engineering and Advanced Computing, Zhengzhou, China. His main research interests include computer architecture, reverse engineering, and security. He has published over 100 refereed papers. He is a senior member of China Computer Federation.



**Yaobin Xie** received MS and BA degrees from the PLA Information Engineering University, China in 2004 and 2007, respectively. He is currently a PhD candidate and a member of China Computer Federation. His main research interests include reverse engineering, industrial control system, and security.

**Wenzhi Chen** received the PhD degree from Zhejiang University in 2005. He is currently a professor and a PhD supervisor with the College of Computer Science and Technology, Zhejiang University. His areas of research include computer graphics, computer architecture, system software, embedded systems, and security. He has published over 80 refereed papers. He has served as an editorial board member of several international journals, including *IEEE Transactions on Information Forensics and Security*. He is a senior member of IEEE and China Computer Federation.



**Zhongyu Lu** received BA degree from Zhejiang University in 2012. He is currently working toward the PhD degree in the College of Computer Science and Technology at Zhejiang University, China. His research interests include computer architecture, cache optimization, and emerging NVM.