# Template for Preparation of Manuscripts for *Tsinghua Science and Technology*

This template is to be used for preparing manuscripts for submission to *Tsinghua Science and Technology*. Use of this template will save time in the review and production processes and will expedite publication. However, use of the template is not a requirement of submission. Do not modify the template in any way (delete spaces, modify font size/line height, etc.).

TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 0?/?? pp???-???
Volume 22, Number 3, June 2017

# A TrustEnclave-based Architecture for Run-time Security on Embedded Terminals

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

**Abstract:** The run-time security guarantee is a hotspot in current cyberspace security research, especially on embedded terminals, e.g., smart hardware, wearable devices, mobile devices. Typically, these devices use universal hardware and software to connect with public network by internet, and are probably open to security threats from Trojan, virus and other malware. As a result, not only personal sensitive data is threatened, economic interests in industry are also 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. Last, both analytical and experimental evaluations are given in the end. The experimental results demonstrate that the proposed security scheme is effective and feasible.

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

## 1 Introduction

Embedded terminals (e.g., smart hardware, wearable devices, mobile devices) have recently attracted lots of attentions in cyberspace security community. On one hand, embedded system has already been the central part of control system and weapon system in military field. On the other hand, sundry embedded devices have been used by several infrastructure control facilities in civil domain, automobile control, industrial control, transportation system, electric system, financial system, mobile communication, and so forth. With the rapid development of Internet of Things technology and the promotion of mobile embedded devices' computation performance, the new pattern informatization application has come into being, for instance, Industrial 4.0, BYOD (Bring Your Own Device), and so on. A lot of attention has been drawn to embedded terminals, e.g. smart devices in enterprise office network, smart hardware, wearable devices, mobile devices. Typically, these devices use universal hardware and software to connect with public network by internet, and are probably open to security threats from Trojan, virus and other malware. As a result, not only personal sensitive data is threatened, economic interests in industry are also 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 theoretics and technologies are not capable of applying to embedded devices' protection mechanism directly. In fact, the security problems of embedded systems are much more complicated than desktop systems. As a matter of fact, how to improve the security of embedded terminals has been an urgent yet challenging problem. As we know,

Type the authors affiliation together with E-mail address here.
- 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

* Kexue Road, NO.62, MailBox403, Zhengzhou, 450001, China
Manuscript received: year-month-day; revised: year-month-day; accepted: year-month-day

security vulnerabilities from OS (Operating System) or third-party software are increasingly serious. It is not an effective way to protect run-time systems by preventing vulnerabilities and patching faulty programs as before. Nowadays, it is lack of mature theories and fundamental researches that focus on run-time security of embedded devices. The mainstream implementation schemes are based on virtualization technology and secure coprocessor. Virtualization technology utilizes 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, VM supervisor owns more vulnerabilities than OS due to its complexity. Furthermore, the performance overhead owing to hardware virtualization is unacceptable for computation capability of embedded devices. Additionally, not all the embedded devices have virtualization support. Kernel vulnerabilities detection scheme based on secure coprocessor has already been proposed [1], but it only supplies isolated execution environment with a lack of controlling capability on system resources, such as memory and other exterior equipment. Such a way would lead to two negative results. One is that monitoring function deployed in kernel's address space is easy to be tampered by attackers. Consequently, the monitoring in real time would fail. The other is that the scheme based on 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 cannot prevent the execution of illegal processes.

In order to address the security problems for embedded terminals effectively, this paper explores several key technologies of operating system support for run-time security (Section 3), 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 OS kernel, which cannot be tampered by the untrusted OS kernel itself. TrustEnclave, 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 in the end (Section 5). The experimental results demonstrate that the proposed security scheme is effective and feasible. It is expected that the proposed architecture and implementation scheme would better support potential applications on embedded terminals where run-time security is desired (e.g., the smart devices).

The main contributions of this paper are:

(1)We explore mainstream technologies recent years and compare existing implement schemes.

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

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

## 2 Overview

Embedded system is a custom-built measurement system with demanding functions, reliability, cost, volume, and power dissipation. It consists of embedded microprocessor, hardware platform, embedded OS, and applications. Embedded system is similar to computer system, which owns 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 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 that are physically separated. One environment contains the main OS and applications, the other environment contains trusted software components. We thus 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, which offers stronger guarantees than an equivalent outside of the security perimeter. However, since the trusted area is separated by hardware from OS and applications, its isolation is guaranteed. 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 (abbreviated to TEE), and the one in the untrusted area Rich Execution Environment (abbreviated to REE). Indeterministic software in the REE cannot affect software running in the TEE.

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

## 3 Background and Motivation

The research on key technologies of operation system 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 privilege compared with OS, the security enhancement scheme based on virtualization technology enhances system security by isolating and monitoring. It usually deploys monitoring tool outside the system. Thus, monitoring tool can't be manipulated by malicious software. Besides, the untrusted software running in special virtual machine, which are likely to be manipulated by malicious software, can't bypass hypervisors and influence other virtual machines. Secvisor utilized SVM (Secure Virture Machine) of AMD processor to supply run-time kernel code integrity protection, which results in much performance overhead and is not portable for embedded system [3]. At present, the virtualization products of mobile embedded terminal field are vmware, L4Android, OKL4, Xen, LXC, etc. 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. Because several virtual machines still shared sole kernel, the insufficience of the scheme was that it didn't supply the solution for kernel attacks.

Depending on hardware and software of current system, TPM is a hardware module which is used to generate and store security key, authenticate digital signature, and produce certificate. LaGrande structure proposed by Intel was an effective solution for both PC system and embedded system [5]. Yu Zheng et al. designed and implemented a scheme for trusted mobile terminal based on hardware platform with OMAP730 processor. Shuyi Chen et al. proposed trusted mobile platform architecture based on MTM (Mobile Trusted Module) and gave the formal verification based on predicate logic [6]. Bo Zhao et al. from Wuhan University designed and implemented a trusted PDA based on chip JetWay2810 [7]. Kim et al. from Korea implemented a highly efficient hardware architecture with SHA-1 and HMAC in 2007, and then in 2010 they designed the first small size MTM chip with triple calculating speed of current TPM and less energy consumption [8, 9].

TPM uses secure key, and anything untrusted didn't know the key. Thus, anything encrypted by the key was considered secure [10]. However, it can't defense 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 legitimate system software to schedule and manage the use of platform resources. It 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) for software security. Because SGX has been the newest security technology of Intel since 2013, how to utilize it on embedded platforms will be full of possibilities and deserve more 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 gave the theoretical analysis for embedded system security with Intel SGX support [12]. Georgia Institute of Technology achieved a project openSGX simulating SGX by QEMU, which is the first attempt to use SGX in embedded field [13].

ARM defines TrustZone [14] as a hardware-supported system-wide approach to security which is integrated in high-performance processors e.g. 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 environment for key system modules and protects system resources in security working mode. Compared to complex hypervisors, TrustZone is a more appropriate method for embedded system security.

Motivated by the above research status, while TrustZone [16] has been introduced more than 10 years, it is only until recently that hardware manufacturers such as 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 make it possible for the research community [17], as well as industry to experiment and develop innovative solutions with TrustZone. This turns towards an opener TrustZone technology.

## 4 Design and implementation

### 4.1 TrustZone-based TEE architecture

In order to support TEE, a device needs to define a security perimeter separated by hardware from the main OS and applications, where only trusted code executes. We show TrustZone-based TEE architecture in Fig.1. We refer to this security perimeter as trusted area called 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 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. This depends 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.

### 4.2 Design challenges

The most powerful feature of TrustZone is that it is capable of securing any peripheral 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 turns on, it will not only prevent

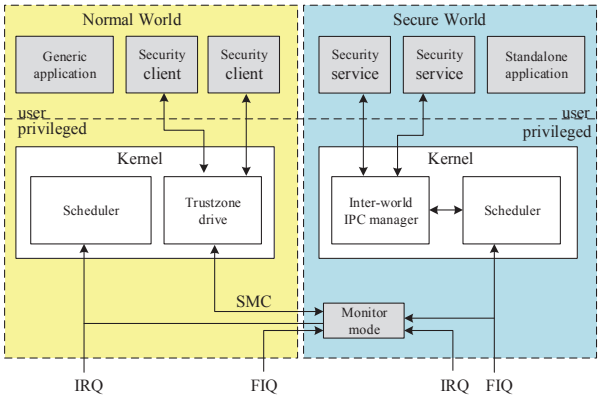



**Fig. 1 TrustZone Architecture**

attacks through 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 the access to hardware, the support from hardware includes two aspects. One is that higher privilege code can't 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 flag bit, we can control the range of physical address space where the privileged code is running. For example, the instruction 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, which can modify crucial state of hardware, only run in the security memory space in plan. Second, the security physical memory space can't be modified. Third, there does not exist such address where it's possible to get a protosomatic sensitive code in the security physical memory space. That means we artificially recode the sensitive code. It is impossible to execute the sensitive code by jumping into security space. The result is twofold. One is that NW can't execute the sensitive code from normal memory because there does not exist sensitive code in the normal memory. The other is that the sensitive code can't execute in NW because NW can't read sensitive code from SW. Recoding can be achieved by two approaches, similar to binary translation in fully-virtualization and kernel modification in para-virtualization. The cost of binary translation for ARM is lower than X86 because of ARM's 32 bits fixed instruction format. When

NW ultimately executes sensitive code, CPU actually executes a SMC call. When NW receives a SMC call, it checks the value of register saved before, which is taken as operation code, and jumps according to protocol. The hardware functions are actualized in SW. Consequently, it comes true that the instructions in NW have the same functions as before and are secure as well.

### 4.3 TrustEnclave-based privilege mode

TrustZone introduces new states of security for ARM architecture, which decide whether 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, ARM processor has seven work modes divided into two categories, i.e., user mode and privileged mode. Access rights to certain resources are restricted in user mode, but they are not constrained in the other six privileged modes.

- User mode: Low-privileged mode, where user code which is outside 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.
- UndefiMd: Illegal instructions being executed.

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

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

**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 |

As shown in Table 2, system call(SVC) and security call(SMC) use the same interrupt vector address. SVC is used to switch user mode to privileged mode, while SMC is used to switch privileged mode to security mode. However, it will cause undefined instruction exception if SMC is called in 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 as follows.

(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 a part of memory space as security space. Access SW only in security state.

(3) Control strictly the entry of entering the SW.

(4) Quit from SW needs to be restricted.

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

If it is in non-security mode, calling system call SMC is the only way to enter security mode. Yet if it is both in user mode and non-security mode, it must call SVC

**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 |

first. It is worthwhile to note that the modification of mode is severely restricted. If it is in non-security mode, calling security call SMC is the only way to change into monitor mode. When it is both in privileged and security mode, we can modify system mode directly. All hardware resources can be accessed in monitor mode.

### 4.4 TrustEnclave Construction

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

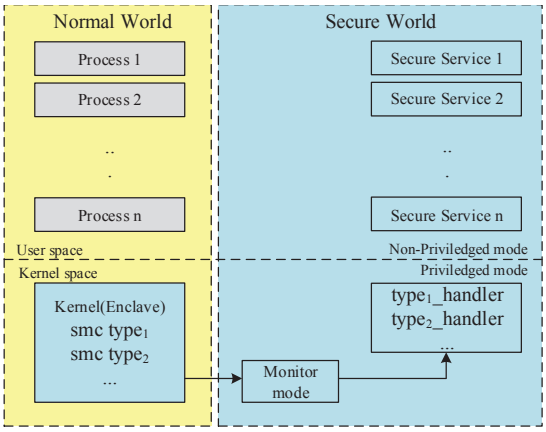



**Fig. 2 The structure of TrustEnclave**

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

In order to assure memory page table mapped read-only, we add new security strategy while switching TTBR (Translation Table Base Register) and updating page table, viz. we must make sure that all physical memory page tables are read-only and writable multimap does not exist. It 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 will be created. One is initializing page table of 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 physical page table will

be created when the first level page table is updated. Both of them 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. Then, 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 three main components:

(1) Trusted operating system which represents a specific way to organize 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;

(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 in order to support innovative services, and this inevitably comes to the cost of exposing components in the trusted area. Still, we will see that 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 sensitive assets.

There are two TrustZone components that are exposed to the untrusted area, therefore are subject to being compromised: the generic TrustZone driver and the secure monitor. These two components are closely related, since 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 had established for them in our design. Indeed, we have satisfied our main objective: increasing the security of embedded system with possible theoretical complex applications but without killing innovation. The untrusted area, where innovative applications execute, can be fully compromised. However, by means of 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 CES-4412P development board which is formed around Samsung newest Exynos4412, viz. a quad-core ARM Cortex-A9 processor. The experimental platform is one of the few platforms fully supporting TrustZone, where TrustZone registers are available. More concretely, we use the CES-4412P development board, which runs typically at 1.4∼1.6GHz with 32KB L1 cache and 1MB L2 cache. The CES-4412P is depicted in Fig.3.

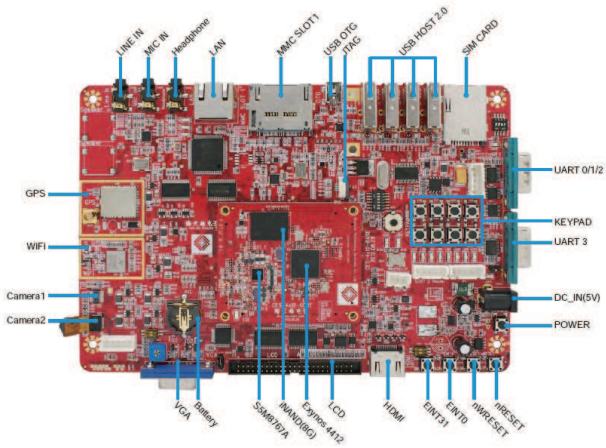



**Fig. 3 Samsung CES-4412P development board**

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 that a secure task (or trusted module) is called from kernel space, a context switch takes place between kernel and secure space. Even though this process is implementation specific, it at least involves: saving the untrusted state, switching software stack, loading the secure state, dispatching the secure task, and returning to kernel space (save secure state, change software stack, load untrusted sate). We denote this double context switch Secure Round Trip. The metric we use is the overhead introduced by the secure space, defined as:

$$Overhead = \frac{\boldsymbol{T}_{secure} - \boldsymbol{T}_{kernel}}{\boldsymbol{T}_{kernel}} \quad (1)$$

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

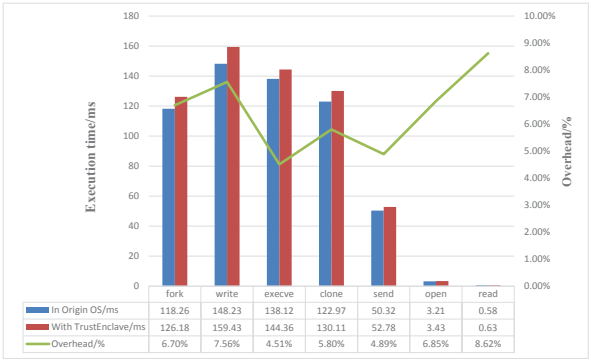



**Fig. 4 Performance evaluation for TrustEnclave**

## 6 Discussion and future work

Different from the schemes based on secure coprocessor, TEE architectures provide processor secure environment, where 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 some form of trigger to allow the core to switch from one state to the other. We implement one of TEE architectures which is different from other international researchers' work. The comparison results are shown in table3.

Based on our experience designing and building support for trusted embedded terminals, we now propose a roadmap for future work. As demonstrated in this work, hardware isolation is indeed an effective solution to provide run-time security in commodity OS without making assumptions on their trustworthiness. Meanwhile, it simply introduces an affordable overhead in terms of performance. On top of our initial hypothesis, our future work include 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.

## 7 Related work

In consequence of size and overhead, the separate TPM chip in embedded terminals is inadequate. TPM module implemented in software is another choice. Aaraj and Raghunathan et al. tested overhead and execution time of software TPM instructions on PDA [18]. Choi et al. from Korea implemented a mobile trusted system based on micro-kernel [19]. Bugiel from Sweden introduced DRTM (Dynamic Root of Trust for Measurement) to protect and measure MTM [20], which tried to establish dynamic trusted computing environment. Researchers Jan-Erik and Markku from Nokia research institute implemented simulator MTM based on simulator TPM [21]. MIPS developed security processor core including extended ISA to accelerate encryption and decryption functions and security memory management. IBM produced a coprocessor distorting authentication. Fengwei Zhang from Georgia Mason University studied one of run-time probable attacks and proposed implantation scheme [22]. Igor Smolyar from Technion aimed at SRIOV [23] utilizing a VM to control another VM.

In recent years, academic researchers focused on ARM-Android platform for embedded terminals security [24], and new technologies and ideas emerged within combination of academia and industry [25]. TrustZone-based technologies applied to mobile terminal field [26], such as Apple SecureEnclave, Samsung KNOX and so forth [27]. After establishing Hypervisor-Based IMA(i.e. HIMA), Professor Azab

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

| Researchers | Secure World | Normal World | Platform | Hardware-assistent | Memory Protection |
|---|---|---|---|---|---|
| S.Pinto [35] | FreeRTOS | Linux | Xilinx ZC702 | Yes | Not-mentioned |
| Javier G [36] | Open Virtualization | Linux3.8.0 | Xilinx ZC702 | Yes | Implement |
| Brian McGilliion [37] | Linux | Android/IOS/Linux | Open-TEE | No | Not-mentioned |
| Johannes Winter [38] | Linux | Android/Linux | QEMU emulator | No | Not-mentioned |
| Xia Yang [39] | T-OS(Trust-E) | Android4.0 | SMDK210 | Yes | Not-mentioned |
| Yingjun Zhang [40] | Open Virtualization | Linux2.6.35 | Xilinx 7000 | Yes | Mentioned |
| Our work | Open Virtualization | Linux4.0.1 | CES-4412P | Yes | Implement |

and Professor Ning Peng from North Carolina State University developed the applications for KNOX and explored some new technologies [28]. Ge X and Vijayakumar H et al. proposed a protection scheme for kernel integrity on mobile embedded devices based on TrustZone without implementation [29]. On .Net platform, a security scheme based on TrustZone and TEE was jointly developed by Microsoft Research and Lisbon University [30]. Researchers from CASED proposed a new code provisioning paradigm for the code intended to run within execution environments established on top of secure hardware [31].

Besides, Ruhr-Universitaet and Microsoft Research Bochum utilized the newest SGX secure mode [32] to isolate physical memory of individual nodes and implemented trustworthy data analytics in the cloud [33]. Seongwook Jin from Korea Advanced Institute of Science and Technology proposed a scheme to monitor hypervisor and protect client resources with hardware assistance [34].

## 8 Conclusion

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

We try 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 operating system support for run-time security become research hotspots. An efficient and feasible implementation scheme is presented. We propose an architecture 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 OS kernel, which cannot be tampered by the untrusted OS kernel itself. TrustEnclave, protected by hardware isolation technology, is an area of OS kernel. Hence, system monitor program should be trusted. Our experiments demonstrate that the proposed security scheme is effective and feasible. It can be used to protect memory, prevent malicious application, insulate sensitive data, and deal with some other problems in the field of embedded system security. It is expected that the proposed architecture and implementation scheme would better support potential applications on embedded terminals where run-time security is desired (e.g., the smart devices).

### Acknowledgements

Many thanks to Xian Chen and Yuxia Cheng for helpful discussion about this work.

Thanks to project supported by the National Natural Science Foundation of China (No.61572516, No.61503213).

### References

[1] Sailer R, Zhang X, Jaeger T, et al. Design and implementation of a TCG-based integrity measurement architecture[C]// Usenix Security Symposium, August 9-13, 2004, San Diego, Ca, Usa. 2004:16-16.

[2] Azab A M, Ning P, Sezer E C, et al. HIMA: A Hypervisor-Based Integrity Measurement Agent[C]// Computer Security Applications Conference. IEEE Computer Society, 2009:461-470.

[3] Seshadri A, Luk M, Qu N, et al. SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes[J]. Sosp, 2007, 41(6):335-350.

[4] Xu L, Chen W, Wang Z. Research about Virtualization of ARM-Based Mobile Smart Devices[J]. Lecture Notes in Electrical Engineering, 2014, 308:259-266.

[5] Barrett M, Thomborson C. USING NGSCB TO MITIGATE EXISTING SOFTWARE THREATS[M]// Certification and Security in Inter-Organizational E-Service. Springer US, 2005:55-74.

[6] Chen S Y, Wen Y Y, Hong Z. Conceptual Design of Trusted Mobile Platform[J]. Journal of Northeastern University, 2008, 29(8):1096-1099.

[7] Bo Z, Guo Z H, Jing L I, et al. The System Architecture and Security Structure of Trusted PDA[J]. Chinese Journal of Computers, 2010, 33(1):82-92.

[8] Kim M, Ju H, Kim Y, et al. Design and implementation of mobile trusted module for trusted mobile computing[J]. IEEE Transactions on Consumer Electronics, 2010, 56(1):134-140.

[9] Teow K S, Dainow E, Nikolaev L, et al. Cryptographic control for mobile storage means: WO, US 8689347 B2[P]. 2014.

[10] Shen D Z, Hu X B, Liu H Z, et al. Security research of state cryptographic authentication security chip in smart grid[C]// China International Conference on Electricity Distribution. IEEE, 2014:416-418.

[11] Hoekstra M, Lal R, Pappachan P, et al. Using innovative instructions to create trustworthy software solutions[C]// International Workshop on Hardware and Architectural Support for Security and Privacy. 2013:1-1.

[12] A. Sadeghi. Trusted Execution Environments Intel SGX. `http://sigops.org/sosp/sosp13/,2014`

[13] Prerit Jain, Soham Desai. Intel SGX Emulation using QEMU. `https://github.com/sslab-gatech/opensgx,2015`

[14] Zhou Y M. The Analysis of TrustZone Secure Technology Based on ARM Architecture[J]. Microcomputer Information, 2008, 24(36):69-71.

[15] Baumann, Andrew, Peinado, Marcus, Hunt, Galen. Shielding applications from an untrusted cloud with Haven[C]// Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2014:1-26.

[16] Alves T. TrustZone : Integrated Hardware and Software Security[J]. White Paper, 2004.

[17] Gonzlez J, Bonnet P. Towards an Open Framework Leveraging a Trusted Execution Environment[C]// the 5th international symposium on cyberspace safety and security. 2013:458-467.

[18] Aaraj N, Raghunathan A, Jha N K. Analysis and design of a hardware/software trusted platform module for embedded systems[J]. Acm Transactions on Embedded Computing Systems, 2008, 8(1):3296-3306.

[19] Choi S G, Han J H, Lee J W, et al. Implementation of a TCG-Based Trusted Computing in Mobile Device[C]// International Conference on Trust, Privacy and Security in Digital Business. Springer-Verlag, 2008:18-27.

[20] Bugiel S, Ekberg J E. Implementing an application-specific credential platform using late-launched mobile trusted module[M]. 2010.

[21] Ekberg J E, Asokan N, Kostiainen K. METHOD AND APPARATUS TO RESET PLATFORM CONFIGURATION REGISTER IN MOBILE TRUSTED MODULE: EP, EP 2537115 A1[P]. 2012.

[22] Zhang F, Leach K, Stavrou A, et al. Using Hardware Features for Increased Debugging Transparency[C]// IEEE Symposium on Security and Privacy. IEEE, 2015:55-69.

[23] Smolyar I, Ben-Yehuda M, Tsafrir D. Securing self-virtualizing ethernet devices[C]// Usenix Conference on Security Symposium. USENIX Association, 2015.

[24] Fahl S, Harbach M, Muders T, et al. Why eve and mallory love android: an analysis of android SSL (in)security[C]// CCS '12: Proceedings of the 2012 ACM conference on Computer and communications security. 2012:50-61.

[25] Kim S H, Han D, Lee D H. Predictability of Android OpenSSL's pseudo random number generator[C]// ACM Sigsac Conference on Computer and Communications Security. 2013:659-668.

[26] Egele M, Brumley D, Fratantonio Y, et al. An empirical study of cryptographic misuse in android applications[C]// ACM Sigsac Conference on Computer and Communications Security. 2013:73-84.

[27] Santos N, Raj H, Saroiu S, et al. Using ARM trustzone to build a trusted language runtime for mobile applications[C]// International Conference on Architectural Support for Programming Languages and Operating Systems. 2014:67-80.

[28] Azab A M, Ning P, Shah J, et al. Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World[C]// ACM Sigsac Conference on Computer and Communications Security. ACM, 2014:1028-1031.

[29] Ge X, Vijayakumar H, Jaeger T. Sprobes: Enforcing Kernel Code Integrity on the TrustZone Architecture[J]. Eprint Arxiv, 2014.

[30] Bugiel S, Ekberg J E. Implementing an application-specific credential platform using late-launched mobile trusted module[M]. 2010.

[31] Dmitrienko A, Heuser S, Nguyen T D, et al. Market-Driven Code Provisioning to Mobile Secure Hardware[M]// Financial Cryptography and Data Security. Springer Berlin Heidelberg, 2015:387-404.

[32] Baumann, Andrew, Peinado, Marcus, Hunt, Galen. Shielding applications from an untrusted cloud with Haven[C]// Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2014:1-26.

[33] F. Schuster, M. Costa, C. Fournet, et al. VC3: Trustworthy Data Analytics in the Cloud Using SGX[J]. IEEE Symposium on Security and Privacy, 2015:38-54.

[34] Jin S, Ahn J, Seol J, et al. H-SVM: Hardware-assisted Secure Virtual Machines under a Vulnerable Hypervisor[J]. IEEE Transactions on Computers, 2015, 64(10):1.

[35] Pinto S, Oliveira D, Pereira J, et al. Towards a lightweight embedded virtualization architecture exploiting ARM TrustZone[C]// IEEE International Conference on Emerging Technologies and Factory Automation. 2014:1-4.

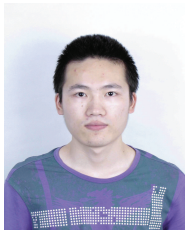
[36] Gonzlez J, Bonnet P. Towards an Open Framework Leveraging a Trusted Execution Environment[C]// Trustdata. 2013:458-467.
[37] Mcgillion B, Dettenborn T, Nyman T, et al. Open-TEE – An Open Virtual Trusted Execution Environment[C]// Trustcom/bigdatase/ispa. IEEE, 2015:58-67.
[38] Winter J, Wiegele P, Pirker M, et al. A Flexible Software Development and Emulation Framework for ARM TrustZone[M]// Trusted Systems. Springer Berlin Heidelberg, 2011:1-15.
[39] Yang X, Shi P, Tian B, et al. Trust-E: A Trusted Embedded Operating System Based on the ARM Trustzone[C]// 2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom). IEEE Computer Society, 2014:495-501.
[40] Zhang Yingjun Feng Dengguo Qin Yu Yang Bo Trusted Computing and Information Assurance Laboratory, Software I O. A Trustzone-Based Trusted Code Execution with Strong Security Requirements[J]. Journal of Computer Research and Development, 2015.

**Rui Chang** Rui Chang was born in 1981. She received the M.S. degree from Wuhan University of Technology in 2007, and B.A. 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 Ph.D. 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** Liehui Jiang was born in 1967. He received M.S. degree in Computer Science and Technology from PLA University of Science and Technology in 1994, the Ph.D. degree and B.A. degree from the PLA Information Engineering University, China in 1989 and 2007, respectively. He is currently a professor and 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.

**Wenzhi Chen** Wenzhi Chen was born in 1969. He received the Ph.D. degree from Zhejiang University in 2005, Hangzhou, China. He is currently a Professor and a Ph.D. 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.

**Yaobin Xie** Yaobin Xie was born in 1981. He received M.S. degree and B.A. degree from the PLA Information Engineering University, China in 2004 and 2007, respectively. He is Ph.D. candidate and member of China Computer Federation. His main research interests include reverse engineering, industrial control system and security.



**Zhongyu Lu** Zhongyong Lu was born in 1992. He received B.A. degree from Zhejiang University in 2012. He is currently working toward the Ph.D. degree in the College of Computer Science and Technology at Zhejiang University, Hangzhou, China. His research interests include computer architecture, cache optimization, and emerging NVM.

.