计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20180522
56(7): 1441-1453, 2019

# 基于边缘计算的可信执行环境研究

宁振宇　张锋巍　施巍松
(韦恩州立大学计算机科学系　美国密歇根州底特律　48202)
(zhenyu.ning@wayne.edu)

# A Study of Using TEE on Edge Computing

Ning Zhenyu, Zhang Fengwei, and Shi Weisong
(*Department of Computer Science, Wayne State University, Detroit, MI, USA* 48202)

**Abstract** The concept of edge computing introduces a new emerging computing model that mitigates the high latency caused by the data transmission in the traditional cloud computing model and helps to keep the privacy-or security-sensitive data confidential. However, the security of the execution environment on the edge nodes is still a non-negligible concern that threatens the whole computing model. Recently, hardware vendors design dedicated trusted execution environments (TEEs) on different platforms, and integrating these TEEs to the edge nodes would be efficient to secure the computation on these nodes. In this paper, we investigate a variety of popular TEEs on the traditional computing model and discuss the pros and cons of each TEE based on recent research. Moreover, we further study two popular TEEs-Intel software guard extensions (SGX) and ARM TrustZone technology, and conduct comprehensive performance and security analysis on an Intel Fog Node Reference Architecture platform and an ARM Juno development board, respectively. The analysis results show that using these hardware-assisted TEEs on edge computing platforms produces low overhead while achieving higher security. The discussion on the security challenges of the TEEs is also presented to help improve the reliability of these TEEs and edge computing.

**Key words** edge computing; trusted execution environments (TEEs); TrustZone; software guard extensions (SGX); system security; fog computing

**摘　要** 边缘计算概念的提出引入了一个新兴的计算模型,它不仅可以缓解传统云计算模型中由于数据传输造成的高延迟问题,同时也有益于保持隐私数据及安全敏感数据的机密性.然而,边缘计算节点本身执行环境的安全性依然是一个不可忽略的问题,它时刻威胁着整个边缘计算模型的安全.得益于硬件厂商在各平台上推出可信执行环境,通过将这些可信执行环境集成至边缘计算节点中可以有效地保障这些节点上运算的安全性.此研究首先分析了一系列传统计算模型中的可信执行环境,并讨论了这些可信执行环境各自的优缺点.其后,在此基础上,深入研究了 Intel 软件防护扩展和 ARM TrustZone 这 2 个流行的可信执行环境,并分别在 Intel 雾计算节点参考设计样机和 ARM Juno 开发板上对这 2 个可信执行环境的安全性和性能进行了分析与测试.结果显示:这些硬件辅助的可信执行环境的引入能够在基本不影响整个系统性能的同时,增强边缘计算平台的安全性.为了帮助提高可信执行环境在边缘计算模型下的可靠性,最后总结了将可信执行环境使用在边缘计算模型中将要面对的安全挑战.

**收稿日期**:2018-07-20;**修回日期**:2019-01-14
**通信作者**:张锋巍(fengwei@wayne.edu)

**关键词** 边缘计算;可信执行环境;TrustZone;软件防护扩展;系统安全;雾计算

**中图法分类号** TP309

云计算(cloud computing)[1]的概念被提出并广泛应用之后,大型应用服务的海量数据计算逐渐转移到云端[2].随着物联网技术(Internet of things, IoT)[3]和万物互联技术(Internet of everything, IoE)[4]的推进,越来越多的节点开始融入互联网,同时也导致部分节点对数据和计算的实时性和隐私性提出了更高的要求.然而由于云计算模型结构要求数据上传,从而导致其无法满足所有节点的要求,尤其是实时性和隐私性要求.因此,边缘计算(edge computing)[5]和雾计算(fog computing)[6]应运而生.

与传统的云计算模型不同的是,边缘计算模型在数据源和云计算中心之间增加了一层网络边缘节点.这些网络边缘节点不仅能够与云计算中心进行双向数据传输,还可以高效地在数据源附近执行部分实时性要求较高的计算任务.由于这些节点更加贴近数据源,他们与数据源的通信成本更低、时效性更高.此外,由于不再需要将数据统一上传到云计算中心,数据的隐私性和安全性也得到了更好的保障.

然而网络边缘节点自身的安全性依然是一个不可忽视的问题.与集中式云计算中心自身的安全性不同,边缘节点的分布式特性决定了难以对其进行统一的管理.因此,如何保证节点计算的安全性和隐私性依然是一个值得探讨的研究问题.

例如在边缘计算视频监控系统中[5],为了降低存储空间并减少数据视频上传,边缘节点将提供视频预处理功能,这也意味着边缘节点拥有所有视频监控的读取权限.这些边缘节点并不像云计算节点一样集中管理,而是分布在各个边缘端,因而它们被恶意入侵的风险更大.一旦这些节点被入侵,所有的监控视频都有可能被恶意读取,进而造成公共隐私泄露等安全问题.又比如在由边缘操作系统支持的智能家居系统中[5],边缘节点集中了整个家居系统中的所有数据,如家庭中照明系统、智能电视、智能监控的数据,并可能拥有其控制权限.一旦这些边缘节点被人恶意控制,将使整个家庭不再有隐私可言,用户甚至将对各种智能家居系统失去控制,造成严重的后果.再比如在由边缘节点支持的智能交通系统中[5],自动驾驶汽车对于汽车的控制基于边缘节点对于周边环境的感知和分析来决策,一旦这些感知和分析出现了问题,自动驾驶汽车可能会做出危险的控制动作,造成严重影响公共安全的毁灭性的后果.这里仅以3个较为典型的场景为例分析了边缘节点被攻击者控制的后果,与传统计算模型一样,边缘计算模型的几乎所有应用场景都不得不面临节点的安全问题[7].

可信执行环境(trusted execution environment, TEE)是指在设备上一个独立于不可信操作系统而存在的可信的、隔离的、独立的执行环境,为不可信环境中的隐私数据和敏感计算提供了一个安全而机密的空间,其安全性通常通过硬件相关的机制来保障.常见的TEE包括Intel软件防护扩展(software guard extensions, SGX)[8-10]、Intel管理引擎(management engine, ME)[11]、x86系统管理模式(system management mode, SMM)[12]、AMD内存加密(memory encryption)技术[13]、AMD平台安全处理器(platform security processor, PSP)[14]和ARM TrustZone技术[15].

近年来,对可信执行环境技术的研究飞速发展,而在边缘节点上应用这一技术可以有效地提高边缘节点的安全性和隐私性.本文将首先对这些技术的背景和概念进行阐述;其次通过2个案例从可信执行环境对性能的影响方面来分析在边缘节点上应用可信执行环境的可行性;最后,我们将讨论可信执行环境所面临的各种挑战.通过本文的阐述,我们期待为边缘计算模型安全性研究者提供未来的研究方向.

在本文的案例分析中,我们使用Intel雾计算样机[16]以及ARM Juno开发板[17]来进行测试.其中,Intel雾计算样机是Intel推出的专门为雾计算设计的计算节点.由于雾计算的概念与边缘计算的概念十分贴近,使用这一样机进行测试能够在一定程度上反映边缘计算节点的真实情况.ARM暂无专门为边缘计算或雾计算设计的样机,但ARM的Juno开发板是唯一由ARM公司推出的开发板,它反映了ARM对嵌入式节点的设计理念,并包含了ARM的高性能处理器以及大量的硬件特性.因此,我们相信它也能代表着ARM对未来雾计算节点或边缘计算节点的设计标准.从这2个平台的性能测试结果来看,Intel软件防护扩展中的模式切换大约需要2.039~2.714 μs,而TrustZone中的模式切换大约需要0.218 μs.从计算的性能来看,Intel软件防护扩展和TrustZone分别引入了1.423倍和1.175倍

额外性能消耗.从 Benchmark 的测试来看,软件防护扩展使单核和多核的性能分别下降了 0.48%和 0.26%,而 TrustZone 则分别下降了 0.13%和 0.02%.因此,整体而言,我们认为在边缘节点中引入可信执行环境对性能的影响是在可接受的范围之内.

## 1 可信执行环境

随着可信执行环境研究的兴起,工业界的各大厂商均进行了研究和尝试,并在各自的软硬件上提供了各种不同的可信执行环境.如表 1 所示,根据实现技术的不同,我们将这些厂商的可信执行环境分为三大类:

1) 基于内存加密的可信执行环境.由于此类解决方案直接在应用层进行隔离,我们认为它的权限级别为 Ring 3;

2) 基于 CPU 模式的可信执行环境.此类解决方案通常使用额外的 CPU 模式来实现内存的隔离,独立于不可信操作系统而存在,因此我们认为它的权限级别为 Ring -2.

3) 基于协处理器(co-processor)的可信执行环境.这一解决方案使用额外的协处理器来提供执行环境,并且这种执行环境有独立的内存,不受主处理器(main processor)和主内存(main memory)的影响,我们认为它的权限级别为 Ring -3.

我们将分别对这 3 种不同类型的可信执行环境进行分析和介绍.

**Table 1 Summary of TEEs**

**表 1 可信执行环境归类**

| Category | Isolated by | Privilege | Example | Related Work |
|---|---|---|---|---|
| Ring 3 TEE | Memory Encryption | Ring 3 privilege, only have access to application memory | Intel Software Guard eXtension<br>AMD Secure Memory Encryption | Ref[18-28] |
| Ring -2 TEE | CPU Modes | Ring -2 privilege, can access all physical memory | x86 System Management Mode<br>ARM TrustZone | Ref[29-53] |
| Ring -3 TEE | Co-processor | Ring -3 privilege, has its own CPU, memory and even power cells | Intel Management Engine<br>AMD Platform Security Processor | Ref[54-57] |

### 1.1 基于内存加密的 Ring 3 可信执行环境

Intel 软件防护扩展[8-10]和 AMD 内存加密技术[58-59]采用内存加密的方式来保护 Ring 3 级别的内存.此类加密技术通过 Ring 3 内存加密来保证在内核不可信的情况下,Ring 3 应用程序的运行环境依旧可信.

#### 1.1.1 Intel 软件防护扩展

2013 年 Intel 公开发表了 3 篇介绍软件防护扩展 SGX 的文章[8-10],正式将这一技术带入研究者的视野.Intel 软件防护扩展实际上是新加入到 Intel 处理器上的一系列扩展指令和内存访问机制.在这一扩展的支持下,应用程序可以在内存中创建 1 个受保护的执行区域,又称围圈(enclave).每一个围圈都可以被视作为 1 个单独的可信执行环境,它们的机密性和完整性由加密的内存来保护.即使基本输入输出系统(basic I/O system, BIOS)、固件(firmware)、管理程序(hypervisors)、操作系统都被恶意修改,软件防护扩展技术依然能保障围圈内执行环境的安全,这一技术也被很多研究学者们认为是 Intel 可信执行技术(trusted execution technology, TXT)[60-61]的接班人.Intel 将软件防护扩展用于支持最新一代的可信计算(trusted computing),并且将在所有后续的 Intel 处理器上集成这一技术来为应用程序提供可信执行环境以解决各种安全问题.

随着 Intel 软件防护扩展的提出,研究学者们提出了一系列基于这一扩展的安全系统.Haven 系统[18]利用软件防护扩展将系统库(system libraries)和库操作系统(library OS)移植进围圈中,利用该围圈来保障其在运行过程中的安全性;Arnautov 等人[19]提出了利用软件防护扩展来为 Docker 提供安全容器(container)的 SCONE 系统,该系统能够有效地帮助容器抵御各种外来攻击;Hunt 等人[20]开发了基于软件防护扩展的分布式沙盒(distributed sandbox)系统 Ryoan,这一系统可以让用户在处理数据时安全地持有他们的数据密钥;Schuster 等人[21]利用软件防护扩展开发了用来为云端 MapReduce 计算提供可信执行环境的 VC3 系统,这一系统为大数据和云计算提供了安全保障;Karande 等人[22]用软件防护扩展来保护系统日志的安全;Shih 等人[23]用软件防护扩展来隔离网络虚拟化(network function virtualization, NFV)应用;SGX-Shield[24]为围圈中的应用程序提供了安全的地址空间布局随机化

(address space layout randomization, ASLR),而T-SGX[25]则用来对抗控制管道攻击,并确保内存缺页(page fault)信息不会被泄露.

### 1.1.2 AMD内存加密技术

AMD在ISCA 2016和USENIX Security 2016会议中介绍了2个x86新特性[58-59]:安全内存加密(secure memory encryption, SME)和安全加密虚拟化(secure encrypted virtualization, SEV).其中安全内存加密定义了一种新的主内存(main memory)加密方式,而安全加密虚拟化则用来与现有的AMD-V虚拟化架构结合以支持加密的虚拟机.这些特性使得选择性地加密部分或者全部系统内存成为可能,并且可以不依赖于管理程序运行加密虚拟机.总体而言,与Intel的软件防护扩展相比,AMD的安全内存加密技术也是1项非常有竞争力的技术——它利用内存加密不仅可以提供Ring 3权限的可信执行环境,也可以提供其他权限级别的可信执行环境(例如管理程序级别,Ring -1权限).同样地,安全加密虚拟化技术能够加密虚拟机,使得运行在虚拟机内的操作系统成为1个可信执行环境.AMD承诺安全内存加密和安全加密虚拟化技术将应用到新的AMD芯片中.AMD最近发布的Zen Processor已经包括了全新的AMD内存加密技术.

## 1.2 基于限制内存访问的Ring -2可信执行环境

Intel系统管理模式[12]和ARM TrustZone技术[15]都采用限制内存访问的方式来创建可信执行环境.具体来说,他们使用硬件来辅助建立内存区域的访问权限,使得系统中的不可信区域的程序无法访问可信区域中的内存.以这种形式创建的可信执行环境中的程序通常与不可信区域中的程序以时间片(time-slice)的形式来共享同一个CPU.

### 1.2.1 x86系统管理模式

系统管理模式[12]始于20世纪90年代早期的Intel奔腾系列CPU,它是x86平台上类似于实模式(real mode)和保护模式(protected mode)的另一种运行模式,AMD随后也在后续的处理器中实现了这一模式.这种运行模式为x86平台特有的电源管理之类的系统控制功能提供了隔离的运行环境.系统管理模式的初始化由基本输入输出系统完成,并通过触发1个系统管理中断(system management interrupt, SMI)来进入.系统管理中断的触发方式有很多种,较为典型的如写入1个硬件端口或用PCI设备生成1个消息信号中断(message signaled interrupt).系统管理中断触发后,CPU将会把其状态信息保存在一段名为系统管理RAM的特殊内存区域中,这一段特殊的内存区域无法被其他执行模式下运行的程序进行寻址与访问.在默认情况下,对系统管理RAM的寻址请求将会被转到图像存储器,这一特性使得系统管理RAM可以被用来当做安全存储设备.值得注意的是,系统管理中断处理程序是由BIOS在系统启动时加载到系统管理RAM中,该处理程序可以不受限制地访问物理内存空间,且可以执行特权指令.因此,系统管理模式也通常被认为拥有Ring -2权限级别.另外,我们可以使用RSM指令来让CPU退出系统管理模式并恢复到之前的模式继续执行.图1显示了x86系统管理模式的基本切换流程.保护模式通过软件或硬件方法触发一个系统管理中断来进入系统管理模式,而系统管理模式中的中断处理器会对这一中断做进一步处理.中断处理完成后,在系统管理模式内使用RSM指令来切换回保护模式.

近年来,基于系统管理模式的研究经常出现在系统安全的学术界研究中.一个典型的例子就是利

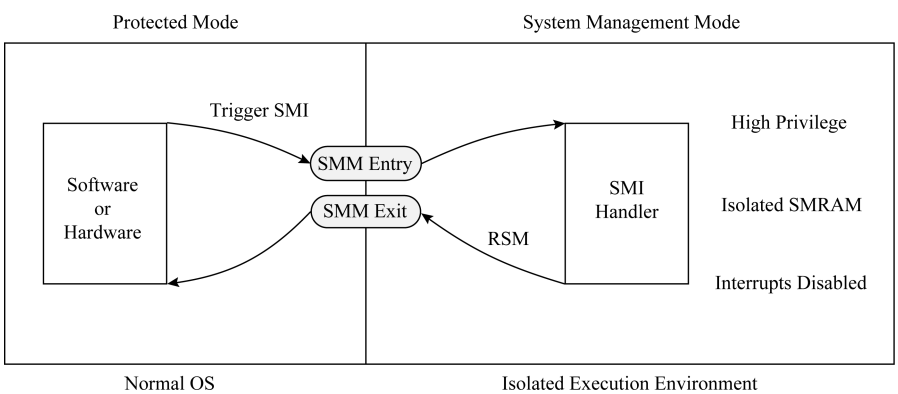


Fig. 1 x86 system management mode

图1 x86系统管理模式

用系统管理模式来检查上层软件(包括管理程序和操作系统)的完整性.HyperGuard[29],HyperCheck[30],HyperSentry[31]都是利用系统管理模式来监控完整性的系统.另外,美国国家科学基金会(National Science Foundation)也在2015年资助了1个利用系统管理模式来在做运行时完整性检测(runtime integrity checking)的项目[32];SICE[33]利用系统管理模式在AMD平台上为敏感计算提供可信执行环境;SPECTRE[34]使用系统管理模式进行实时的内存自省(memory introspection),并以此为基础来进行恶意软件检测;Reina等人[35]以及Jiang等人[36]都使用系统管理模式来可靠地获取系统物理内存以进行取证分析(forensic analysis);IOCheck[37]使用系统管理模式来保护输入输出(I/O)设备的固件和设置;HRA[38]则通过系统管理模式来保护云端的资源审核,使得即使在管理程序被恶意修改后仍能保障资源审核程序的正确性;MalT[39]则更多关注于如何利用系统管理模式来进行裸机(bare-metal)程序调试,并提供更高的透明性(transparency)而不被恶意软件察觉;TrustLogin[40]保护用户的证书及密钥等信息不被不可信运行环境中的软件截获;HOPS[41]则使用系统管理模式来减少自省过程中可被恶意进程侦测的痕迹.

#### 1.2.2 ARM TrustZone技术

ARM TrustZone技术[15]是ARM公司在2002年前后基于ARMv6架构[62]提出的一种硬件新特性,它通过特殊的CPU模式提供了一个独立的运行环境.与其他硬件隔离技术类似,TrustZone将整个系统的运行环境划分为可信执行环境和富运行环境(rich execution environment, REE)两个部分,并通过硬件的安全扩展来确保2个运行环境在处理器、内存和外设上的完全隔离.

如图2所示,支持TrustZone技术的ARM处理器有2种不同的处理模式:普通模式和安全模式.2种处理器模式都拥有各自的内存区域和权限.在普通模式下运行的代码无法访问安全模式拥有的内存区域,而安全模式下运行的程序则可以访问普通模式下的内存.当前处理器所处的模式可以用安全配置寄存器(secure configuration register, SCR)的NS位来判断,而这个寄存器的值只能在安全模式下进行修改.图2同时还描述了ARMv8架构下不同的异常等级(exception level, EL),低的异常等级拥有更低的权限.最高的异常等级是异常级别3,它类似于一个在普通模式和安全模式之间切换的看门人,所有的切换都必须经过异常级别3.普通模式可以通过安全监控调用(secure monitor call, SMC)指令触发1个异常级别3的异常或是触发1个安全中断来进入安全模式,而安全模式则可以通过异常返回(exception return, ERET)指令来返回普通模式.同时,TrustZone利用内存管理单元(memory management unit, MMU)同时在普通模式和安全模式下支持虚拟内存地址.2个模式下同一个虚拟地址空间可以被映射到不同的物理内存区域.在硬件中断方面,ARM支持中断请求(interrupt request, IRQ)和快速中断请求(fast interrupt request, FIQ).通过安全配置寄存器的IRQ位和FIQ位可以将这2种中断分别设置为安全中断,无论普通模式如何进行配置,安全中断都将直接被转送到异常级别3.通常情况下,ARM推荐将中断请求做为普通模式下的中断源,而将快速中断请求做为安全模式下的中断源.

由于现有的移动设备大都采用ARM处理器,研究学者们也利用TrustZone技术对移动设备的安全性进行了大量的研究.TrustDump[42]利用

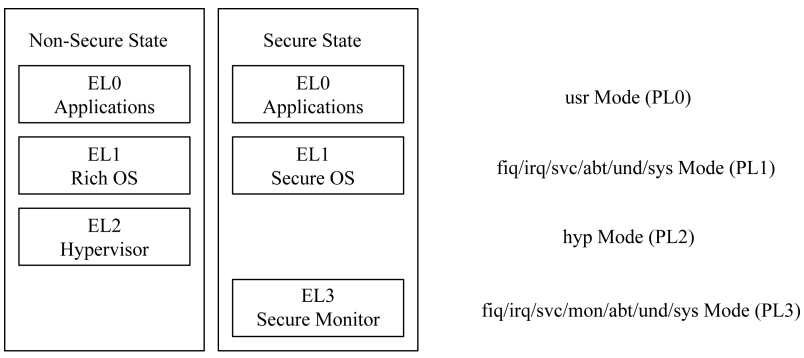


Fig. 2 ARM TrustZone technology

图2 ARM TrustZone技术

TrustZone 来进行可靠的内存信息转储,它通过将 1 个非可屏蔽中断(non-maskable interrupt, NMI)设置为安全中断来完成处理器模式转换,并从安全模式下转储普通模式下的内存;TZ-RKP 系统[43]在安全模式下通过事件驱动的监控来保护普通模式下的操作系统;Sprobes[44]则利用安全模式来审查普通模式下的操作系统,并保障普通模式下内核代码的完整性;SeCReT 系统[45]在普通模式和安全模式之间建立 1 条安全的通信渠道;TrustICE[46]通过 TrustZone 为系统中的敏感计算提供了 1 个安全而隔离的运行环境;TrustOTP[47]提供了 1 个在移动设备上保护一次性密码的解决方案;AdAttester[48]提出了安全的可验证的在线广告架构;Brasser[49]等人则建议利用 TrustZone 来将如摄像头之类的外设控制在受限区域内;fTPM[50]在 TrustZone 中实现了固件版本的可信平台模块(trusted platform module, TPM);PrivateZone[51]利用 TrustZone 来创建一个独立于富运行环境和可信执行环境的独占执行环境;C-FLAT[52]系统利用 TrustZone 中的实时控制流(control-flow)监控来检测控制流劫持;TrustShadow[53]利用 TrustZone 为安全性要求较高的应用程序提供了一个可信执行环境,并将这些应用程序对于操作系统相关服务的请求转送至普通模式下的操作系统,并检查对于每一个请求的回复.

### 1.3 通过协处理器实现的 Ring -3 可信执行环境

与 1.1 节、1.2 节提到的可信执行环境不同的是,Intel 管理引擎[11]和 AMD 平台安全处理器[14]通过 1 个额外的专用协处理器来引入 1 个可信执行环境.协处理器通常与主处理器隔离,并拥有独立的内存和寄存器,因而为这类可信执行环境提供了更有力的安全保障.

#### 1.3.1 Intel 管理引擎

Intel 管理引擎[11]是 1 个嵌入在最新发布的 Intel 处理器内的微型计算机,它存在于 Intel 的服务器、工作站、个人电脑、平板电脑以及手机设备中.Intel 最早提出管理引擎这一概念是在 2007 年,那时该引擎的主要作用是为了支持 Intel 的主动管理技术(active management technology, AMT)[63],而此技术也成为了在 Intel 管理引擎中运行的第 1 个应用程序.最近,Intel 转而利用管理引擎来为安全敏感的应用程序提供可信执行环境.依照 Intel 管理引擎的最新声明[11],有不少的安全应用(如增强的隐私识别、受保护的视频音频路径、身份辨识保护技术、启动保护等)已经或即将在管理引擎中运行.

图 3 所示为 Intel 管理引擎的硬件架构.从图 3 中我们可以发现,这个管理引擎就像 1 台独立的电脑,它有专用处理器、加密引擎、直接内存存取(direct memory access, DMA)引擎、主机嵌入式通讯接口(host-embedded communication interface, HECI)引擎、只读存储器(read-only memory, ROM)、内部静态随机存取存储器(internal static random-access memory, SRAM)、中断控制器、计时器以及其他的输入输出设备.管理引擎的指令在其处理器上执行,而内部静态随机存取存储器则用来保存固件代码和实时数据.管理引擎的处理器也同时拥有代码缓存和数据缓存,可以减少对内部静态随机存取存储器的访问次数.除内部静态随机存取存储器外,管理引擎也会使用一部分主机系统的动态随机存取存储器(dynamic random-access memory, DRAM),这一部分的动态随机存取存储器就像是管理引擎的硬盘空间.管理引擎中处理器暂时不用的代码和数据内存页会被从静态随机存取存储器中移除并换出到主机内存的动态随机存取存储器中.要注意的是,这一段动态随机存取存储器区域在系统启动时就由 BIOS 保留下来专门给管理引擎使用,主机的操作系统无法访问到这一区域.

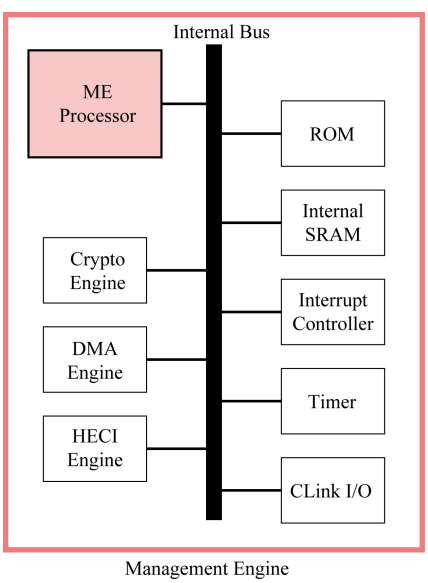


Fig. 3 Intel management engine

图 3 Intel 管理引擎

#### 1.3.2 AMD 台安全处理器

与 Intel 的管理引擎类似,AMD 平台安全处理器[14]也是 1 个嵌入在在 AMD 主处理器内的专用处理器.这个专用处理器利用 ARM 的 TrustZone 技术和 Ring -2 级别的可信执行环境来运行可信的

第三方应用程序.通过平台安全处理器,AMD 保障了从 BIOS 到可信执行环境的安全启动,而受保护的可信第三方应用程序则可以利用工业级别的 API 来使用可信执行环境.另外,系统管理单元(system management unit, SMU)[64]是北桥上在系统启动和运行时用来负责一系列系统和电源管理任务的子部件,它的内部也包含了 1 个处理器[65].而由于 AMD 将北桥直接集成至处理器中,系统管理单元中的这个处理器也可以被视作为一个主处理器中的嵌入式协处理器.

## 2 案例分析

本节中,我们将对应用最广泛的 Intel 软件防护扩展和 ARM TrustZone 这 2 个典型可信执行环境的实际应用进行分析,以阐述在边缘节点中应用这些可信执行环境的可行性.由于硬件的支持已经保障了可信执行环境的隔离性和安全性,而边缘计算节点对性能较为敏感,我们将重点关注 Intel 软件防护扩展和 ARM TrustZone 引入造成的额外性能成本.

### 2.1 Intel 软件防护扩展

Intel 于 2017 年 5 月推出了根据雾计算参考设计的真机样品[16],其配备了 8 核 Intel Xeon E3-1275 处理器、32 GB DDR4 内存、M.2 PCIE 固态硬盘以及 SATA 固态硬盘,软件方面则是使用 Tianocore 开源 BIOS 和 64 位 Ubuntu 16.04 操作系统.由于雾计算的概念与边缘计算类似,且真机样品的处理器包含了对 Intel 软件防护扩展的支持,我们将该样品机作为边缘计算节点,测试 Intel 软件防护扩展的性能.

在雾计算真机样品软件环境的基础上,我们部署了 Intel 软件防护扩展软件开发 SDK 1.9[66]来支持我们的测试.之后我们将测试 Intel 软件防护扩展在进行模式切换时的时间消耗、在围圈模式中进行计算导致的额外时间成本,以及围圈模式内的计算对整个系统的性能影响.

#### 2.1.1 模式切换时间

为了计算模式切换所需要的时间,我们利用 Intel 提供的 API 在围圈内定义了 1 个空函数.从围圈外调用该函数时,CPU 将会切换为围圈模式,而该函数执行完毕并返回后,CPU 将退出围圈模式.为了精确地计算切换所需的时间,我们使用 RDTSC 指令来读取 CPU 周期(cycle),并通过对比 CPU 周期数量以及对应的 CPU 频率来计算围圈模式切换所花费的时间.由于围圈模式下禁用该指令,我们无法准确测量从非围圈模式进入围圈模式或是退出非围圈模式所需的时间,而只能测量从非围圈模式进入围圈模式,再回到非围圈模式这一完整周期所需要的时间.另外,2 种模式之间的参数传递由指定大小的缓存实现,而这一缓存的大小也会对模式切换的性能产生影响,因此我们使用不同大小的缓存来测量 Intel 软件防护扩展中模式切换的时间.为了减少 CPU 频率调节(frequency scaling)所带来的误差,我们将 CPU 频率固定为 4 GHz,并且将这一实验重复 1 000 次.

我们的实验结果如表 2 所示,从表 2 中我们可以发现,在不使用缓存传递参数的情况下,1 次完整的围圈模式切换需要的时间大约为 2.039 μs.而随着缓存的增大,模式切换所需要的时间也同步增加.当缓存大小分别 1 KB,4 KB,8 KB,16 KB 时,模式切换所需的时间分别增加到 2.109 μs,2.251 μs,2.362 μs,2.714 μs.通过表 2 中这一切换时间的 95%可信区间不难看出,这一切换时间的可信区间(confidence interval, CI)范围非常小,也就说明了我们的测试结果十分稳定.

**Table 2 Switching Time of SGX Enclave**

**表 2 围圈模式切换时间**

| Cache Size/KB | Switch Time/μs | | |
|---|---|---|---|
| | Mean | STD | 95% CI |
| 0 | 2.039 | 0.066 | [2.035,2.044] |
| 1 | 2.109 | 0.032 | [2.107,2.111] |
| 4 | 2.251 | 0.059 | [2.247,2.254] |
| 8 | 2.362 | 0.055 | [2.359,2.366] |
| 16 | 2.714 | 0.036 | [2.712,2.716] |

#### 2.1.2 MD5 散列计算时间

在这一阶段的实验中,我们使用开源的基于 RFC 1321 标准的 MD5 算法[67]来计算字符串的散列值.我们分别在围圈模式和非围圈模式下应用该算法计算同一段预先生成的长度为 1 024 的随机字符串的散列值.我们先测量非围圈模式下进行此散列计算的时间消耗,随后在非围圈模式下调用围圈模式中的函数来进行这一计算,并比较这 2 次计算的时间消耗.与 2.1.1 节中的实验类似,我们同样将这一实验重复 1 000 次来减少误差.

从表 3 中可以发现,在非围圈模式下进行该 MD5 散列计算所需的时间约为 4.734 μs,而调用围圈模式中的函数则需要约 6.737 μs,这 2 种计算的时间差距约为 2.003 μs.值得注意的是,本节实验中在围圈模式下进行 MD5 计算的时间应为进行围圈模式切换的时间加上围圈模式下进行 MD5 计算的

时间.而在 2.1.1 节的实验中我们测出的围圈模式切换时间也约为 2 μs,与本节实验中 2 种方式进行 MD5 计算的时间差基本吻合,这也说明在围圈模式和非围圈模式中 CPU 的性能基本相当,在围圈模式中进行计算引入的额外性能消耗即为围圈模式切换所需的性能消耗.

**Table 3 Time Consumption of MD5**

**表 3 MD5 计算时间**

| Mode | Switch Time/μs | | |
|---|---|---|---|
| | Mean | STD | 95% CI |
| Normal | 6.737 | 0.081 | [6.732,6.742] |
| Enclave | 4.734 | 0.095 | [4.728,4.740] |

2.1.3 整体系统性能影响

为了测试 Intel 软件防护扩展中运行在围圈模式下的代码对整个系统性能的影响,我们使用一个每隔 1 s 进行 1 次模式切换的应用程序来模拟系统中的敏感计算.该程序每次切换到围圈模式后,会进行一次 1 024 长度字符串的 MD5 散列计算,随后切换回非围圈模式.通过比较启动该程序前后整个系统的性能,我们可以得知该计算对整体性能的影响.为了准确测量多核 CPU 系统的性能,我们使用 GeekBench 4.1.1[68] 的 Linux 版本来测试我们的 Intel 雾计算真机样品的性能分,此实验被重复了 100 次来减少随机误差.

表 4 显示了在有无敏感计算的情况下 GeekBench 给出的雾计算真机样品的性能分.在没有敏感计算的情况下,单核性能分约为 4 327.325;而存在敏感计算的情况下,单核性能分约为 4 306.458,性能降低了约 0.48%.同样地,在没有敏感计算的情况下,多核性能分约为 17 739.62,而存在敏感计算的情况下多核性能分约为 17 692.72,性能降低了约 0.26%.很明显,即使是每秒钟 1 次的频繁模式切换,对整体系统性能的影响也几乎可以忽略不计.

**Table 4 Performance Score with GeekBench**

**表 4 GeekBench 测试性能分**

| Mode | Single-Core | | Multi-Core | |
|---|---|---|---|---|
| | Mean | STD | Mean | STD |
| Normal | 4 327.325 | 17.124 | 17 739.62 | 114.598 |
| Enclave | 4 306.458 | 14.850 | 17 692.72 | 110.244 |

## 2.2 ARM TrustZone

我们在 ARM 平台上使用 ARM Juno v1 开发板[16]进行实验.该开发板上的处理器包含了 1 组双核 Cortex-A57 处理器集群和 1 组 4 核 Cortex-A53 处理器集群.存储方面,Juno 开发板搭载了 8 GB DDR3L 内存以及 64 MB NOR 闪存.由于 ARM 尚未推出专门用于边缘计算或是雾计算的硬件,我们把 Juno 开发板当做边缘计算中的一个节点,并在 ARM 可信固件(ARM trusted firmware) v1.1[69] 和 Linaro 发布的安卓 5.1.1 版本交付件[70]的基础上进行本阶段的实验.与针对 Intel 软件防护扩展的测试类似,我们将分析 TrustZone 进行模式切换的时间消耗、在安全模式中进行计算导致的额外时间成本,以及安全模式中计算对整个系统的性能影响.

2.2.1 模式切换时间

由于指令集的差异,ARM 架构中不支持使用 RDTSC 指令来读取 CPU 周期.在 Juno 开发板所搭载的 Cortex-A53 和 Cortex-A57 处理器中,我们可以使用处理器附带的性能监视单元(performance monitor unit, PMU)来计算 2 个时间点之间所经过的 CPU 周期数.我们在执行触发模式切换的指令前打开性能监视单元,并记录当前 CPU 周期数,然后在进入到安全模式时以及回到普通模式后分别记录 CPU 周期数.通过这些记录值的差距,我们可以计算出从非安全模式进入安全模式、从安全模式进入非安全模式,以及 1 次完整的从非安全模式进入安全模式再回到非安全模式的切换所需要的时间.与 Intel 软件防护扩展不同的是,在 TrustZone 的安全模式和非安全模式之间传递参数并不需要通过缓存,而是通过寄存器直接传递,因而不会带来任何额外的性能消耗,所以这里的实验不再测试不同大小的参数值对模式切换产生的影响.在本节的实验中,我们将 CPU 频率设置为 1.15 GHz,并重复实验 1 000 次以减少误差.

表 5 显示了本阶段的实验结果.我们可以发现,从非安全模式进入安全模式大概需要 0.135 μs,而从安全模式退回非安全模式大概需要 0.082 μs,1 次完整的从普通模式到安全模式再回到普通模式的切换所需要的时间约为 0.218 μs.我们注意到 1 000 次实验结果之间的差异很小,这说明 TrustZone 中的模式切换时间基本不受外在因素的干扰.

**Table 5 Time Consumption of Mode Switch**

**表 5 模式切换时间**

| Operation | Switch Time/μs | | |
|---|---|---|---|
| | Mean | STD | 95% CI |
| Enter Secure Mode | 0.135 | 0.001 | [0.135,0.135] |
| Quit Secure Mode | 0.082 | 0.003 | [0.082,0.083] |
| Overall | 0.218 | 0.005 | [0.218,0.219] |

### 2.2.2 MD5 散列计算时间

在本节的实验中,我们使用与 2.1.2 节中同样的开源算法代码以及随机字符串来进行 MD5 散列计算.我们自定义了一个内核模块(kernel module),并分别测量在该模块中直接进行散列计算和使用安全监控调用来切换到安全模式下进行散列计算的时间消耗.同样,该组实验重复 1 000 次以减少误差.

本阶段的实验结果如表 6 所示.从表 6 可见,在普通模式的内核模块中直接进行散列计算需要的时间约为 8.229 μs,而切换到安全模式下进行散列计算所需的时间约为 9.670 μs,两者之间的差距为 1.441 μs.我们注意到 2.2.1 节中测出的 1 次完整模式切换所需的时间仅为 0.346 μs,因此,我们猜测 CPU 在安全模式下的计算性能有略微的降低.

**Table 6 Time Consumption of MD5**

**表 6 MD5 计算时间**

| Mode | Switch Time/μs | | |
|---|---|---|---|
| | Mean | STD | 95% CI |
| Normal Mode | 8.229 | 0.231 | [8.215,8.244] |
| Secure Mode | 9.670 | 0.171 | [9.660,9.681] |

### 2.2.3 整体系统性能影响

与 2.1.3 节中的测试类似,我们编写了一个安卓环境下的 Linux 可执行程序来模拟边缘节点中的敏感数据计算,该程序每隔 1 s 会进行 1 次从非安全模式到安全模式的切换.在安全模式下,边缘节点会进行 1 次 2.2.2 节中的字符串 MD5 计算,随后退回到非安全模式.为了了解在这一频繁的敏感计算对整个节点性能的影响,我们通过谷歌应用商店中的 GeekBench 4 应用来测试此程序执行时和不执行时边缘节点的整体性能分.与 Linux 版本的 GeekBench 类似,安卓环境下的 GeekBench 同时对单核的性能以及多核的整体性能进行测试估分.值得注意的是,为了减少随机误差,我们重复此实验 100 次.

从表 7 中可以看出,在有无敏感计算的情况下单核平均分分别为 983.440 和 984.700,而多核平均分分别为 2 143.920 和 2 144.420.频繁的模式切换和敏感计算使单核性能分和多核性能分分别下降了 0.13%和 0.02%,这说明模式切换引入的性能下降基本是可以忽略的.

**Table 7 Performance Score with GeekBench**

**表 7 GeekBench 测试性能分**

| Mode | Single-Core | | Multi-Core | |
|---|---|---|---|---|
| | Mean | STD | Mean | STD |
| Normal Mode | 984.700 | 1.878 | 2 144.42 | 5.560 |
| Secure Mode | 983.440 | 3.273 | 2 143.92 | 6.386 |

## 2.3 性能比较

综合本节的所有实验,我们可以看出 ARM TrustZone 中的模式切换比 Intel 软件防护扩展中的模式切换要快 5.89 倍左右.然而单就散列计算而言,Intel 雾计算真机样品的计算速度更快,这也可能是与 CPU 频率有关.Intel 雾计算真机样品中的 CPU 频率能达到 4 GHz,而 ARM Juno 开发板上的 CPU 频率最高只能达到 1.15 GHz.从对系统整体性能的影响来看,软件防护扩展和 TrustZone 对整个系统的影响都可以忽略不计.

图 4 总结了 Intel 与 ARM 在 GeekBench 测试中的得分.从图 4 中不难看出,Intel 架构下的系统比 ARM 架构下的系统性能更占优势,这也是由于

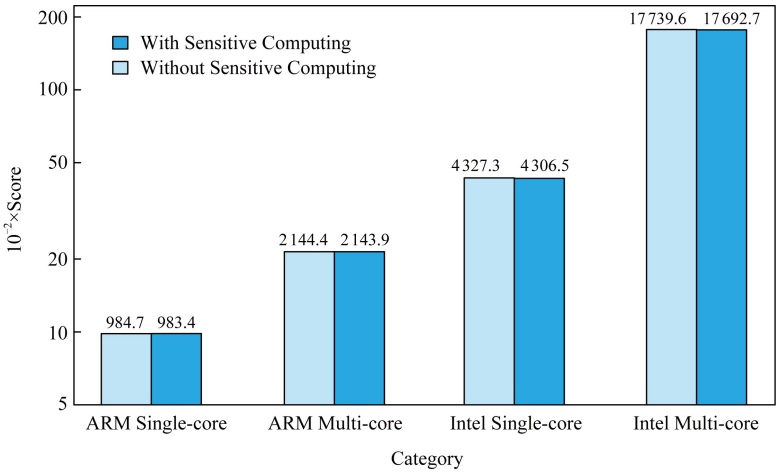


Fig. 4 Performance comparison on Intel and ARM platforms

图 4 Intel 和 ARM 平台的性能比较

ARM 系统设计的初衷是为了在移动端能够以最少的能耗做最多的计算,而 Intel 系统在设计中并不需要考虑能耗问题.在边缘节点的设计中,我们可以根据不同的需求来选择不同的架构.若可以保证该节点的持续供电,我们可以选用性能更强大的 Intel 架构,反之,我们则可以选用能耗较低的 ARM 架构.

## 3 挑战与未来

虽然现有的可信执行环境已经可以应用在边缘节点来提高节点的安全性,但这些可信执行环境自身仍然存在一些缺陷和漏洞,可能会影响整个边缘计算模型的安全.

Schwarz 等人[26]通过缓存侧信道(side channel)来攻击 Intel 软件防护扩展,并演示了如何在几分钟获取围圈中内保存的以 mbedTLS 实现的 RSA 加密私钥.Brasser 等人[27]则演示了一种更有效的攻击,它通过软件防护扩展中的缓存信息泄漏来攻击人类基因组索引项目;AsyncShock[28]说明了可以通过攻击来控制线程调度,并利用这一控制来寻找围圈中的同步漏洞.

在 x86 系统管理模式中,SMRAM 由 BIOS 锁定,系统启动后其他任何处理器模式下都不能访问 SMRAM.然而,Rutkowska 和 Wojtczuk 分别通过内存回收(memory reclaiming)[29]以及缓存投毒(cache poisoning)技术[71]绕过这一锁定来访问 SMRAM;Duflot 等人[72]也整理了系统管理模式的一些设计漏洞;Wojtczuk 等人[73]展示了通过操控 UEFI 启动脚本来绕过系统管理模式的锁定,从而允许从 Ring 0 权限下修改系统管理中断处理器;Butterworth 等人[74]也发现了系统管理模式中 BIOS 升级的一个缓冲区溢出漏洞.

Shen[75]利用 ret2usr 攻击获取系统中的 root 权限,随后利用一个边界检查漏洞来实现往任意物理内存地址中写入 1B 的数据,这也最终导致了可以在 TrustZone 的安全模式下执行任意代码.Rosenberg[76]在高通(Qualcomm)基于 Trust-Zone 实现的高通安全执行环境(Qualcomm's secure execution environment, QSEE)中通过一个整数溢出漏洞来实现对安全模式下任意内存的写入,而重写 TrustZone 中的安全系统调用处理器则导致可以在安全模式下执行任意代码;ARMageddon[77]使用 Prime+Probe 缓存攻击来泄露安全模式下的信息,从而实现了在普通模式下监控安全模式下的代码执行.

Tereshkin 等人[54]通过在 Intel 主动管理技术中注入代码来在 Intel 管理引擎中实现了一个 Ring 3 权限的 rootkit;DAGGER[55]使用类似的技术破坏了 Intel 管理引擎中的隔离性.与 Tereshkin 等人使用的技术不同的是,DAGGER 是通过在管理引擎固件的 memset 函数上使用钩子(hook)完成攻击,因为这一函数调用更加频繁.最近,Intel 自身也公开了管理引擎中主动管理技术中的一个漏洞 CVE-2017-5689,即 INTEL-SA-00075[56],攻击者可以通过这一漏洞在 Intel 机器上不用输入密码而获取管理员权限[7].

所有的这些攻击,都说明了可信执行环境自身的安全性仍然需要进一步完善.随着这些可信执行环境的功能逐渐完善,在可信环境的执行代码也会进一步增加,这就导致了这些可信执行环境的可信计算基础(trusted computing base, TCB)逐步增大.而可信计算基础的增大则不可避免地引入了更多可能的漏洞.

## 4 总 结

在大数据的驱动下,传统的云计算模式已经无法有效地解决实时性、高额负载以及数据的安全隐私问题.伴随着物联网以及万物互联的推进,边缘计算更加凸显其重要性.

本文主要从数据安全的角度,阐述怎么样在边缘计算的背景下建立一个独立的、隔离的、安全的可信执行环境.我们首先总结了现有硬件支持的可信执行环境;随后,我们对 2 种应用广泛的可信执行环境进行测试分析,在此测试分析中,我们采用了 Intel 雾计算的节点的模拟真机以及 ARM 公司自己的开发板,以增强我们分析的结果在边缘计算模型下的实际性和代表性.

总体来说,我们的分析和实验表明,将现有的可信执行环境融入到边缘计算模型当中可以有效地提高边缘计算节点中关键计算的安全性和可信度.与此同时,可信执行环境的引入对边缘节点的性能影响也较小.因此,这一方案是切实可行的.

我们希望通过本文能够引起产业界和学术界对可信执行环境在边缘计算中重要性的关注.我们相信,在不久的将来,会有越来越多的可信执行环境应用到边缘计算中,使得边缘计算更加地安全和可靠.

## 参 考 文 献


[1] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing [J]. Communications of the ACM, 2010, 53(4): 50-58

[2] Dinh H T, Lee C, Niyato D, et al. A survey of mobile cloud computing: Architecture, applications, and approaches [J]. Wireless Communication and Mobile Computing, 2013, 13(18): 1587-1611

[3] Gubbi J, Buyya R, Marusic S, et al. Internet of things (IoT): A vision, architectural elements, and future directions [J]. Future Generation Computer Systems, 2013, 29(7): 1645-1660

[4] Etzion O, Fournier F, Arcushin S, et al. Tutorial on the Internet of everything [C] //Proc of the 8th ACM Int Conf on Distributed Event-Based Systems. New York: ACM, 2014: 236-237

[5] Shi Weisong, Cao Jie, Zhang Quan, et al. Edge computing: Vision and challenges [J]. IEEE Internet of Things Journal, 2016, 3(5): 637-646

[6] Bonomi F, Milito R, Zhu Jiang, et al. Fog computing and its role in the Internet of things [C] //Proc of the 1st Edition of the MCC Workshop on Mobile Cloud Computing. New York: ACM, 2012: 13-16

[7] Deng Xiaoheng, Guan Peiyuan, Wan Zhiwen, et al. Integrated trust based resource cooperation in edge computing [J]. Journal of Computer Research and Development, 2018, 55(3): 449-477 (in Chinese)
(邓晓衡, 关培源, 万志文, 等. 基于综合信任的边缘计算资源协同研究[J]. 计算机研究与发展, 2018, 55(3): 449-477)

[8] Anati I, Gueron S, Johnson S, et al. Innovative technology for CPU based attestation and sealing [OL]. [2017-05-03]. https://software.intel.com/en-us/articles/innovative-technology-for-cpu-based-attestation-and-sealing

[9] Hoekstra M, Lal R, Pappachan P, et al. Using innovative instructions to create trustworthy software solutions [OL]. [2017-05-03]. https://software.intel.com/en-us/articles/using-innovative-instructions-to-create-trustworthy-software-solutions

[10] Mckeen F, Alexandrovich I, Berenzon A, et al. Innovative instructions and software model for isolated execution [C] //Proc of the 2nd Int Workshop on Hardware and Architectural support for Security and Privacy. New York: ACM, 2013: Article No.10

[11] Ruan Xiaoyu. Platform Embedded Security Technology Revealed: Safeguarding the Future of Computing with Intel Embedded Security and Management Engine [M]. Berkeley, CA: Apress, 2014

[12] Intel. Intel® 64 and IA-32 architectures software developer manuals [OL]. [2018-05-18]. https://software.intel.com/en-us/articles/intel-sdm

[13] Kaplan D, Powell J, Woller T. AMD memory encryption [OL]. [2016-06-19]. http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf

[14] AMD. AMD secure technology [OL]. [2017-06-18]. http://www.amd.com/en-us/innovations/software-technologies/security

[15] ARM. Security technology: Building a secure system using TrustZone [OL]. [2015-10-07]. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

[16] Intel. Fog reference design overview [OL]. [2017-09-16]. https://www.intel.com/content/dam/www/public/us/en/documents/design-guides/fog-reference-design-overview-guide.pdf

[17] ARM. Juno ARM development platform [OL]. [2016-04-15]. https://developer.arm.com/products/system-design/development-boards/juno-development-board

[18] Baumann A, Peinado M, Hunt G. Shielding applications from an untrusted cloud with Haven [C] //Proc of the 11th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2014: 267-283

[19] Arnautov S, Trach B, Gregor F, et al. SCONE: Secure Linux containers with Intel SGX [C] //Proc of the 12th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2016: 689-703

[20] Hunt T, Zhu Zhiting, Xu Yuanzhong, et al. Ryoan: A distributed sandbox for untrusted computation on secret data [C] //Proc of the 12th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2016: 533-549

[21] Schuster F, Costa M, Fournet C, et al. VC3: Trustworthy data analytics in the cloud using SGX [C] //Proc of the 36th IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2015: 38-54

[22] Karande V, Bauman E, Lin Zhiqiang, et al. SGX-Log: Securing system logs with SGX [C] //Proc of the 12th ACM Asia Conf on Computer and Communications Security. New York: ACM, 2017: 19-30

[23] Shih M W, Kumar M, Kim T, et al. S-NFV: Securing NFV states by using SGX [C] //Proc of the 2016 ACM Int Workshop on Security in Software Defined Networks and Network Function Virtualization. New York: ACM, 2016: 45-48

[24] Seo J, Lee B, Kim S. SGX-Shield: Enabling address space layout randomization for SGX programs [C] //Proc of the 24th Annual Network and Distributed System Security Symp. Reston, VA: Internet Society, 2017

[25] Shih M W, Lee, S, Kim T, et al. T-SGX: Eradicating controlled-channel attacks against enclave programs [C] //Proc of the 24th Annual Network and Distributed System Security Symp. Reston, VA: Internet Society, 2017

[26] Schwarz M, Weiser S, Gruss D, et al. Malware guard extension: Using SGX to conceal cache attacks [C] //Proc of the 14th Conf on Detection of Intrusions and Malware and Vulnerability Assessment. Berlin: Springer, 2017: 3-24

[27] Brasser F, Muller U, Dmitrienko A, et al. Software grand exposure: SGX cache attacks are practical [C] //Proc of the 11th USENIX Workshop on Offensive Technologies. Berkeley, CA: USENIX Association, 2017

[28] Weichbrodt N, Kurmus A, Pietzuch P, et al. AsyncShock: Exploiting synchronisation bugs in intel SGX enclaves [C] //Proc of the 21st European Symp on Research in Computer Security. Berlin: Springer, 2016: 440-457

[29] Rutkowska J, Wojtczuk R. Preventing and detecting Xen hypervisor subversions [OL]. [2015-10-24]. https://invisiblethingslab.com/resources/bh08/part2-full.pdf

[30] Zhang Fengwei, Jiang Wang, Sun Kun, et al. Hypercheck: A hardware-assistedintegrity monitor [J]. IEEE Transactions on Dependable and Secure Computing. 2014, 11(4): 332-344

[31] Azab A M, Ning Peng, Wang Zhi, et al. Hypersentry: Enabling stealthy in-context measurement of hypervisor integrity [C] //Proc of the 17th ACM Conf on Computer and Communications Security. New York: ACM, 2010: 38-49

[32] Karavanic K. TWC: Small: System infrastructure for SMM-based runtime integrity measurement [OL]. [2016-07-25]. https://nsf.gov/awardsearch/showAward?AWD_ID=1528185

[33] Azab A M, Ning Peng, Zhang Xiaolan. Sice: A hardware-level strongly isolated computing environment for x86 multi-core platforms [C] //Proc of the 18th ACM Conf on Computer and Communications Security. New York: ACM, 2011: 375-388

[34] Zhang Fengwei, Leach K, Sun Kun, et al. Spectre: A dependable introspection framework via system management mode [C] //Proc of the 43rd IEEE/IFIP Int Conf on Dependable Systems and Networks. Piscataway, NJ: IEEE, 2013: 1-12

[35] Reina A, Fattori A, Pagani F, et al. When hardware meets software: A bulletproof solution to forensic memory acquisition [C] //Proc of the 28th Annual Computer Security Applications Conf. New York: ACM, 2012: 79-99

[36] Jiang Wang, Zhang Fengwei, Sun Kun, et al. Firmware-assisted memory acquisition and analysis tools for digital forensics [C] //Proc of the 6th IEEE Int Workshop on Systematic Approaches to Digital Forensic Engineering. Piscataway, NJ: IEEE, 2011

[37] Zhang Fengwei. IOCheck: A framework to enhance the security of I/O devices at runtime [C] //Proc of the 43rd Annual IEEE/IFIP Conf on Dependable Systems and Networks Workshop. Piscataway, NJ: IEEE, 2013

[38] Jin S, Seol J, Huh J, et al. Hardware-assisted secure resource accounting under a vulnerable hypervisor [J]. ACM SIGPLAN Notices, 2015, 50(7): 201-213

[39] Zhang Fengwei, Leach K, Stavrou A, et al. Using hardware features for increased debugging transparency [C] //Proc of the 36th IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2015: 55-69

[40] Zhang Fengwei, Leach K, Wang Haining, et al. TrustLogin: Securing password-login on commodity operating systems [C] //Proc of the 10th ACM Symp on Information, Computer and Communications Security. New York: ACM, 2015: 333-344

[41] Leach K, Spensky C, Weimer W, et al. Towards transparent introspection [C] //Proc of the IEEE 23rd Int Conf on Software Analysis, Evolution, and Reengineering. Piscataway, NJ: IEEE, 2016: 248-259

[42] Sun He, Sun Kun, Wang Yuewu, et al. TrustDump: Reliable memory acquisition on smartphones [C] //Proc of the 19th European Symp on Research in Computer Security. Berlin: Springer, 2014: 202-218

[43] Azab A M, Ning Peng, Shah J, et al. Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world [C] //Proc of the 21st ACM Conf on Computer and Communications Security. New York: ACM, 2014: 90-102

[44] Ge Xinyang, Vijayakumar H, Jaeger T. Sprobes: Enforcing kernel code integrity on the trustzone architecture [C] //Proc of the 3rd IEEE Mobile Security Technologies Workshop. Piscataway, NJ: IEEE, 2014

[45] Jang JS, Kong S, Kim M, et al. SeCReT: Secure channel between rich execution environment and trusted execution environment [C] //Proc of the 22nd Annual Network and Distributed System Security Symp. Reston, VA: Internet Society, 2015

[46] Sun He, Sun Kun, Wang Yuewu, et al. TrustICE: Hardware-assisted isolated computing environments on mobile devices [C] //Proc of the 45 Annual IEEE/IFIP Int Conf on Dependable Systems and Networks. Piscataway, NJ: IEEE, 2015: 367-378

[47] Sun He, Sun Kun, Wang Yuewu, et al. TrustOTP: Transforming smartphones into secure one-time password tokens [C] //Proc of the 22nd ACM Conf on Computer and Communications Security. New York: ACM, 2015: 976-988

[48] Li Wenhao, Li Haibo, Chen Haibo, et al. AdAttester: Secure online mobile advertisement attestation using trustzone [C] //Proc of the 13th Annual Int Conf on Mobile Systems, Applications, and Services. New York: ACM, 2015: 75-88

[49] Brasser F, Kim D, Liebchen C, et all. Regulating arm TrustZone devices in restricted spaces [C] //Proc of the 14th Annual Int Conf on Mobile Systems, Applications, and Services. New York: ACM, 2016: 413-425

[50] Raj H, Saroiu S, Wolman A, et al. fTPM: A software-only implementation of a TPM chip [C] //Proc of the 25th USENIX Security Symp. Berkeley, CA: USENIX Association, 2016: 841-856

[51] Jang J, Choi C, Lee J, et al. PrivateZone: Providing a private execution environment using ARM TrustZone [J]. IEEE Transactions on Dependable and Secure Computing, 2016, 15(5): 797-810

[52] Abera T, Asokan N, Davi L, et al. C-FLAT: Control-flow attestation for embedded systems software [C] //Proc of the 23rd ACM Conf on Computer and Communications Security. New York: ACM, 2016: 743-754

[53] Guan Le, Liu Peng, Xing Xinyu, et al. TrustShadow: Secure execution of unmodified applications with ARM TrustZone [C] //Proc of the 15th Annual Int Conf on Mobile Systems, Applications, and Services. New York: ACM, 2017: 488-501

[54] Tereshkin A, Wojtczuk R. Introducing ring-3 rootkits [C/OL] //Proc of 2009 Black Hat USA. [2016-02-14]. https://invisiblethingslab.com/resources/bh09usa/Ring%20-3%20Rootkits.pdf

[55] Patrick S, Iurii B. Understanding DMA Malware [C] //Proc of the 9th Conf on Detection of Intrusions and Malware & Vulnerability Assessment. Berlin: Springer, 2012: 21-41

[56] Intel. Intel-SA-00075 [OL]. [2018-03-21]. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00075.html

[57] Carlos Perez. Tenable Blog: Rediscovering the Intel AMT Vulnerability [OL]. [2018-04-15]. https://www.tenable.com/blog/rediscovering-the-intel-amt-vulnerability

[58] Kaplan D. AMD x86 memory encryption technologies [OL]. [2016-10-18]. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kaplan

[59] Kaplan D, Woller T, Powell J. AMD memory encryption tutorial [OL]. [2016-07-28]. https://sites.google.com/site/metisca2016/

[60] Costan V, Devadas S. Intel SGX explained [OL]. [2017-12-15]. https://eprint.iacr.org/2016/086.pdf

[61] Rutkowska J. Intel x86 considered harmful [OL]. [2015-10-29]. https://media.8ch.net/cyber/src/1457971916926-2.pdf

[62] Brash D. The ARM architecture version 6 [OL]. [2016-05-30]. http://lars.nocrew.org/computers/processors/ARM/ARMv6.pdf

[63] Intel. Intel® active management technology [OL]. [2018-03-15]. https://www.intel.com/content/www/us/en/architecture-and-technology/intel-active-management-technology.html

[64] Marek R. AMD x86 SMU firmware analysis-Do you care about Matroshka processors [OL]. [2017-05-26]. https://events.ccc.de/congress/2014/Fahrplan/system/attachments/2503/original/ccc-final.pdf

[65] AMD. BIOS and kernel developer's guide (BKDG) for AMD family 15h models 00h-0Fh pocessors [OL]. [2017-10-24]. http://support.amd.com/TechDocs/52740_16h_Models_30h-3Fh_BKDG.pdf

[66] Intel. Intel® Software Guard Extensions (Intel® SGX) SDK [OL]. [2017-12-04]. https://software.intel.com/en-us/sgx-sdk/download

[67] Rivest R. The MD5 message-digest algorithm [OL]. [2018-03-21]. https://www.ietf.org/rfc/rfc1321.txt

[68] Primate Labs. GeekBench 4 [OL]. [2018-04-12]. https://www.primatelabs.com/

[69] ARM. Trusted Firmware [OL]. [2017-10-23]. https://github.com/ARM-software/arm-trusted-firmware

[70] Linaro. Linaro Releases [OL]. [2016-12-24]. http://releases.linaro.org/android/reference-lcr/juno/15.09/

[71] Wojtczuk R, Rutkowska J. Attacking SMM memory via Intel CPU cache poisoning [OL]. [2015-08-16]. http://composter.com.ua/documents/Attacking-SMM-Memory-via-Intel-CPU-Cache-Poisoning.pdf

[72] Duflot L, Levillain O, Morin B, et al. Getting into the SMRAM: SMM reloaded [C/OL] //Proc of the 19th Annual CanSecWest Conf. [2017-05-21]. https://cansecwest.com/csw09/csw09-duflot.pdf

[73] Wojtczuk R, Kallenberg C. Attacking UEFI boot script [C/OL] //Prof of the 31st Chaos Communication Congress. [2017-05-22]. https://papers.put.as/papers/firmware/2014/venamis_whitepaper.pdf

[74] Butterworth J, Kallenberg C, Kovah X, et al. Bios chronomancy: Fixing the core root of trust for measurement [C] //Proc of the 8th ACM Conf on Computer and Communications Security. New York: ACM, 2013: 25-36

[75] Shen Di. Attacking your trusted core: Exploiting trustzone on Android [C] //Proc of 2015 Black Hat USA. [2016-04-09]. https://www.blackhat.com/docs/us-15/materials/us-15-Shen-Attacking-Your-Trusted-Core-Exploiting-Trustzone-On-Android.pdf

[76] Rosenberg D. Reflections on trusting trustzone [OL]. [2016-03-01]. https://www.blackhat.com/docs/us-14/materials/us-14-Rosenberg-Reflections-on-Trusting-TrustZone.pdf

[77] Lipp M, Gruss D, Spreitzer R, et al. ARMageddon: Cache attacks on mobile devices [C] //Proc of the 25th USENIX Security Symp. Berkeley, CA: USENIX Association, 2016: 549-564

**Ning Zhenyu**, born in 1986. PhD candidate at Wayne State University, MI, USA. His main research interests include edge computing, hardware-assisted security, trusted execution environment and transparent debugging.

**Zhang Fengwei**, born in 1986. Assistant professor, PhD supervisor. COMPASS Lab Director, Wayne State University, MI, USA. His main research interests include system security, hardware-assisted security, trustworthy execution environments, tracing and debugging transparency.

**Shi Weisong**, born in 1974. Professor, PhD supervisor. IEEE Fellow, Charles H. Gershenson Distinguished Faculty Fellow, ACM Distinguished Scientist. His main research interests include edge computing, computer system, and energy-efficiency.