(19)国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 118673503 A
(43)申请公布日 2024.09.20

(21)申请号 202410823308.X

(22)申请日 2024.06.24

(71)申请人 南方科技大学
地址 518055 广东省深圳市南山区学苑大道1088号
申请人 支付宝(杭州)信息技术有限公司

(72)发明人 张锋巍 刘念 宁振宇 罗嘉俊 朱嘉楠 闫守孟

(74)专利代理机构 北京亿腾知识产权代理事务所(普通合伙) 11309
专利代理师 陈霁 周良玉

(51)Int.Cl.
G06F 21/57(2013.01)
G06F 21/53(2013.01)
G06F 21/55(2013.01)

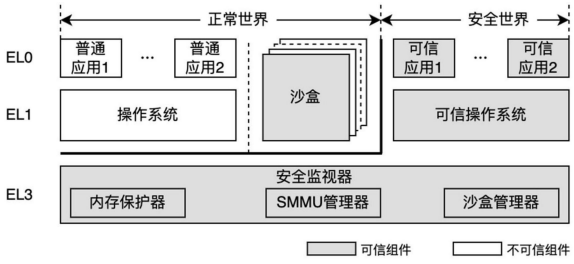
权利要求书2页 说明书9页 附图2页

(54)发明名称

一种基于可信区架构提供可信执行环境的方法和设备

(57)摘要

本说明书实施例提供一种基于可信区架构提供可信执行环境的方法和设备。可信区架构包括,安全世界和非安全世界;其中非安全世界中运行有主机操作系统OS,安全世界运行有安全监视器,安全监视器包括,内存保护器和沙盒管理器。方法包括:响应于沙盒创建请求,主机OS为第一沙盒分配第一内存区段。然后,内存保护器对运行主机OS的CPU核所使用的第一页表进行第一更新,使得更新后的第一页表中不包含对所述第一内存区段的映射。内存保护器还创建第二页表,并将其设置为运行所述第一沙盒的CPU核使用的页表;其中所述第二页表中仅包含对所述第一内存区段的映射。沙盒管理器启动第一沙盒,运行第三方部署在其中的目标应用,为目标应用提供可信执行环境。



CN 118673503 A

1.一种基于可信区架构提供可信执行环境的方法,所述可信区架构包括,安全世界和非安全世界;其中非安全世界中运行有主机操作系统OS,安全世界运行有安全监视器,所述安全监视器包括,内存保护器和沙盒管理器,所述方法包括:

响应于沙盒创建请求,所述主机OS为第一沙盒分配第一内存区段;

所述内存保护器对运行主机OS的CPU核所使用的第一页表进行第一更新,使得更新后的第一页表中不包含对所述第一内存区段的映射;

所述内存保护器创建第二页表,并将其设置为运行所述第一沙盒的CPU核使用的页表;其中所述第二页表中仅包含对所述第一内存区段的映射;

所述沙盒管理器启动所述第一沙盒,运行第三方部署在其中的目标应用,为目标应用提供可信执行环境。

2.根据权利要求1所述的方法,其中,所述安全监视器还包括SMMU管理器;所述方法还包括:

响应于第一沙盒请求使用第一外设,所述内存保护器更新第二页表,使得更新后的第二页表包含所述第一外设对应的内存映射MMIO区段中的映射;

所述SMMU管理器对所述第一外设对应的SMMU页表进行更新,使得SMMU页表中仅包含所述第一内存区段的映射。

3.根据权利要求2所述的方法,在所述第一沙盒请求使用第一外设之前或之时,所述第一外设由所述主机OS使用;所述方法还包括:

内存管理器对所述第一页表进行第二更新,所述第二更新包括,清除所述第一页表中所述第一外设对应的MMIO区段中的映射;

所述SMMU管理器对所述第一外设对应的SMMU页表进行更新,具体包括,从所述SMMU页表中移除所述主机OS使用的内存段的映射,并添加所述第一内存区段的映射。

4.根据权利要求1所述的方法,其中,所述第一页表和第二页表为二阶段地址转译机制中的第二阶段页表。

5.根据权利要求1所述的方法,其中,所述内存保护器创建第二页表,并将其设置为运行所述第一沙盒的CPU核使用的页表,包括:

内存管理器设置运行所述第一沙盒的CPU核中的目标寄存器,使其存储所述第二页表在内存中的起始地址。

6.根据权利要求1所述的方法,其中,在启动所述第一沙盒之前,所述方法还包括:

所述沙盒管理器初始化所述第一沙盒的元数据,其中包括,沙盒编号,第一内存区段的物理地址,第二页表的基地址。

7.根据权利要求1所述的方法,其中,所述主机OS内核中包括用于创建沙盒的创建内核模块,所述沙盒创建请求通过调用所述创建内核模块而发出。

8.根据权利要求1所述的方法,其中,所述主机OS内核中包括用于结束沙盒的结束内核模块;所述方法还包括:

响应于目标应用执行完成,所述结束内核模块调用沙盒管理器,清除第一沙盒的沙盒数据;

所述主机OS释放所述第一内存区段。

9.根据权利要求8所述的方法,其中,清除第一沙盒的沙盒数据,包括:

沙盒管理器清除所述第一内存区段中的数据,销毁第一沙盒的元数据。

10.一种基于可信区架构提供可信执行环境的装置,所述可信区架构包括,安全世界和非安全世界;所述装置包括,运行于非安全世界的主机操作系统OS,运行于安全世界的安全监视器,所述安全监视器包括内存保护器,沙盒管理器,其中:

所述主机OS配置为,响应于沙盒创建请求,为第一沙盒分配第一内存区段;

所述内存保护器配置为,对运行主机OS的CPU核所使用的第一页表进行第一更新,使得更新后的第一页表中不包含对所述第一内存区段的映射;

所述内存保护器还配置为,创建第二页表,并将其设置为运行所述第一沙盒的CPU核使用的页表;其中所述第二页表中仅包含对所述第一内存区段的映射;

所述沙盒管理器配置为,启动所述第一沙盒,运行第三方部署在其中的目标应用,为目标应用提供可信执行环境。

11.根据权利要求10所述的装置,其中,所述安全监视器还包括SMMU管理器;

所述内存保护器还配置为,响应于第一沙盒请求使用第一外设,更新第二页表,使得更新后的第二页表包含所述第一外设对应的内存映射MMIO区段中的映射;

所述SMMU管理器配置为,对所述第一外设对应的SMMU页表进行更新,使得SMMU页表中仅包含所述第一内存区段的映射。

12.根据权利要求11所述的装置,在所述第一沙盒请求使用第一外设之前或之时,所述第一外设由所述主机OS使用;

所述内存管理器还配置为,对所述第一页表进行第二更新,所述第二更新包括,清除所述第一页表中所述第一外设对应的MMIO区段中的映射;

所述SMMU管理器具体配置为,从所述SMMU页表中移除所述主机OS使用的内存段的映射,并添加所述第一内存区段的映射。

13.一种计算设备,包括存储器和若干处理器,所述存储器中存储有可执行代码,所述处理器执行所述可执行代码时,实现权利要求1-9中任一项所述的方法。

一种基于可信区架构提供可信执行环境的方法和设备

技术领域

[0001] 本说明书一个或多个实施例涉及可信区技术,尤其涉及一种在可信区架构中提供可信执行环境的方法、装置和设备。

背景技术

[0002] 由于信息技术的迅速发展和普及,当今社会进入了大数据时代,网络空间中的数据信息每天都在呈爆炸式增长。需要加强数据保护和隐私保护的措施,以促进数据的良性流通和开放共享。为了同时保证数据的可流动性和安全性,提出了可信执行环境的概念。可信执行环境(Trusted Execution Envi ronment,TEE)通过基于硬件的加密或隔离在中央处理器CPU中构建一个安全的区域,保护其中的程序和数据的机密性和完整性,防止程序的执行过程被恶意软件或攻击者所干扰和破坏。它能够提供一种安全的计算环境,保护敏感数据不被泄露或篡改,同时也可以有效地防止黑客攻击和恶意软件的入侵。可信执行环境能够提供硬件级别的安全保障,使数据处理更加安全和可靠。

[0003] 目前,使用较广的商业可信执行环境解决方案主要有Intel的Software Guard Extensions(SGX)和ARM的TrustZone,以及AMD的Secure Encrypted Vi rtual ization (SEV)。这三种可信执行环境技术都拥有自己独特的隔离技术和安全特性,并且都已经被应用于各种场景中。其中,ARM TrustZone即可信区技术,主要应用于移动设备和嵌入式系统领域。市面上绝大部分智能手机和平板电脑都采用了ARM架构的处理器,并且在智能家居、智能穿戴、工业自动化等领域,ARM架构处理器也得到了广泛的应用。

[0004] 虽然TrustZone为应用程序提供了有助于提高安全性的可信执行环境,以及一些基本的安全服务,但大多数应用程序实际上无法利用TrustZone的安全特性,因为TrustZone对第三方应用程序开发者来说有许多使用限制。TrustZone是为供应商而不是第三方应用程序开发者设计的。如果应用程序开发人员想把他们的敏感代码放入TrustZone,他们必须与供应商合作。并且需要开发人员有足够的前置知识并且需要投入大量的开发精力,否则就会产生漏洞,导致隔离环境的安全性被破坏。

[0005] 应用程序的迅猛发展重塑了安全领域对可信执行环境的需求,可信执行环境的研究趋势逐渐从以前由供应商控制的封闭式可信执行环境过渡到开放式可信执行环境,这种可信执行环境可以托管来自多个来源且具有独立任务的可信应用程序。2019年,ARM提出了机密计算架构(Confident ia l Comput ing Arch itecture,CCA),通过为开发者提供隔离的领域(Rea lm)来解决这个问题。然而,CCA需要使用复杂的虚拟化技术来隔离Rea lm,这会导致大量的性能和内存开销,从而限制了它在资源受限的移动和嵌入式设备中的可用性。此外,CCA尚未在现实世界中得到广泛部署,规范的不成熟和新旧软硬件之间的兼容性问题导致CCA的广泛应用可能需要数年时间。因此,亟需更加高效和轻量级的解决方案,来满足第三方开发者对于移动和嵌入式设备的安全需求。

发明内容

[0006] 本说明书一个或多个实施例描述了一种基于可信区架构提供可信执行环境的方法及装置,能够基于已有的可信区架构,高效灵活地为第三方应用提供可信执行环境。

[0007] 根据第一方面,提供一种基于可信区架构提供可信执行环境的方法,所述可信区架构包括,安全世界和非安全世界;其中非安全世界中运行有主机操作系统OS,安全世界运行有安全监视器,所述安全监视器包括,内存保护器和沙盒管理器,所述方法包括:

[0008] 响应于沙盒创建请求,所述主机OS为第一沙盒分配第一内存区段;

[0009] 所述内存保护器对运行主机OS的CPU核所使用的第一页表进行第一更新,使得更新后的第一页表中不包含对所述第一内存区段的映射;

[0010] 所述内存保护器创建第二页表,并将其设置为运行所述第一沙盒的CPU核使用的页表;其中所述第二页表中仅包含对所述第一内存区段的映射;

[0011] 所述沙盒管理器启动所述第一沙盒,运行第三方部署在其中的目标应用,为目标应用提供可信执行环境。

[0012] 根据第二方面,提供了一种基于可信区架构提供可信执行环境的装置,所述可信区架构包括,安全世界和非安全世界;所述装置包括,运行于非安全世界的主机操作系统OS,运行于安全世界的安全监视器,所述安全监视器包括内存保护器,沙盒管理器,其中:

[0013] 所述主机OS配置为,响应于沙盒创建请求,为第一沙盒分配第一内存区段;

[0014] 所述内存保护器配置为,对运行主机OS的CPU核所使用的第一页表进行第一更新,使得更新后的第一页表中不包含对所述第一内存区段的映射;

[0015] 所述内存保护器还配置为,创建第二页表,并将其设置为运行所述第一沙盒的CPU核使用的页表;其中所述第二页表中仅包含对所述第一内存区段的映射;

[0016] 所述沙盒管理器配置为,启动所述第一沙盒,运行第三方部署在其中的目标应用,为目标应用提供可信执行环境。

[0017] 根据第三方面,提供了一种计算设备,包括存储器和若干处理器,所述存储器中存储有可执行代码,所述处理器执行所述可执行代码时,实现第一方面的方法。

[0018] 在本说明书实施例提供的方案中,在正常世界中创建若干沙盒,用于部署第三方应用。通过安全监视器中的内存保护器为运行不同环境的不同CPU核提供不同的内存页表视图,强制沙盒与主机操作系统隔离执行。关于外设的使用,SMMU管理器通过将SMMU的管理转移到安全监视器,确保主机操作系统无法利用外设发起DMA攻击,内存保护器通过页表设置确保不同环境对外设的独占访问权。如此,以轻量级的沙盒环境的方式,为第三方应用提供可信执行环境。

附图说明

[0019] 为了更清楚地说明本发明实施例的技术方案,下面将对实施例描述中所需要使用的附图作简单地介绍,显而易见地,下面描述中的附图仅仅是本发明的一些实施例,对于本领域普通技术人员来讲,在不付出创造性劳动的前提下,还可以根据这些附图获得其它的附图。

[0020] 图1示出根据一个实施例的基于可信区架构创建的系统的结构示意图;

[0021] 图2示出根据一个实施例的基于可信区架构提供可信执行环境的方法;

[0022] 图3示出在一个实施例中各个CPU核的内存页表视图;

[0023] 图4示出根据一个实施例的沙盒生命周期过程示意。

具体实施方式

[0024] 下面结合附图,对本说明书提供的方案进行描述。

[0025] 当前,Armv8平台所使用的TrustZone可信区技术虽然可以为厂商提供提前嵌入可信应用的服务,但是无法将这一服务开放给第三方开发者使用。而面向开发者提供可信执行环境服务的Armv9机密计算架构仍然需要时间来落地。

[0026] 现有的Armv8可信执行环境工作中,为了给第三方开发者提供服务,主要采用了两种隔离手段。一种是基于虚拟化的虚拟机方案(例如vTZ、Twinvi sor、Pr ivateZone等),虽然它们通过虚拟化技术提供了有效的隔离环境,但是虚拟机监视器的引入带来了额外的工作负载,并且虚拟机监视器的攻击面较大,一旦一个虚拟机攻破了虚拟机监视器,就可以读取整个内存。第二种是基于访问控制的隔离方案。例如Trust ICE通过数字水印来保护隔离环境的内存,但是无法在隔离环境运行的同时保证原有操作系统的正常运行。因此,总体来说,目前广泛使用的Armv8可信区架构中,难以为第三方应用提供灵活的、轻量级的可信执行环境。

[0027] 为解决这一问题,本说明书提供一种技术构思及其实施方式,其中,基于可信区架构,在非安全世界中创建轻量级的沙盒环境,用于部署第三方开发者开发的第三方应用。通过安全世界中的安全监视器确保沙盒之间,以及沙盒与原有操作系统的隔离。从而,利用沙盒为第三方应用提供可信执行环境,确保其执行过程的安全。

[0028] 为了描述上述技术构思的实施,首先对可信区架构及其中的一些重要组件进行介绍。

[0029] 一、可信区技术架构

[0030] TrustZone可信区技术是Arm平台上实现可信执行环境TEE的安全机制。可信区架构引入了安全世界(Secure Wor ld)和非安全世界(Norma l Wor ld,或称为正常世界),从而为不同安全要求的代码和数据创建隔离的环境。非安全世界用于不受信的软件栈,而安全世界运行可信应用。启用TrustZone技术的处理器所执行的指令可以基于特权级划分为4个异常等级,EL0至EL3,其中,EL0表示应用级,EL1用于操作系统OS。EL2是可选的,表示虚拟机管理器(hypervi sor),EL3运行Arm可信固件ATF。这四个等级也可用于表示运行环境的权限等级。在TrustZone可信区技术中,CPU安全状态被划分为非安全(Non-Secure)状态和安全(Secure)状态。从EL0到EL2,物理CPU均可以在两种安全状态之间切换,从而以不同的硬件隔离方式执行不同的任务。例如,可以在非安全世界的EL1中执行非可信的操作系统OS(untrusted OS),在安全世界的EL1中执行可信OS。而EL3永远持续运行于安全世界,并通过安全监视器调用SMC(Secure Mon itor Ca l l)提供全面的安全相关服务。安全监视器SM(Secure Mon itor)是EL3中的可信软件,用于处理所有的SMC调用,并具有配置安全相关硬件的权限。

[0031] Arm架构中一个重要的硬件组件为地址空间控制器TZASC(TrustZone Address Space Contro l ler)。TZASC作用为总线管理器和内存芯片之间的过滤器,来确保对内存的正确、合法访问。TZASC将内存划分为安全区域和非安全区域,其中对安全内存区域的访

问被限制为安全状态,而对非安全内存区域的访问可以在任何状态下进行。近来,Arm还在TZASC中引入了基于身份的过滤机制。利用这样的机制,TZASC为各外设群组分配不同的非安全访问标识NSAID(Non-Secure Access Ident it ies),并将内存划分为不同的区域。该机制要求,划分的各个区域必须是连续的,并且区域数量受限(共计9个区域)。安全监视器SM可以针对每个NSAID配置可变的访问权限。

[0032] 二、系统内存管理单元SMMU

[0033] 本领域技术人员可以理解,内存管理单元MMU是内存与CPU之间的硬件组件,CPU使用MMU进行内存访问。当主机连接有外围设备(简称外设),外设对内存的访问控制则需要用到系统内存管理单元SMMU。

[0034] SMMU是用于管理外设对主机内存的直接内存访问DMA的硬件部件。DMA允许外设直接向主机内存传输数据,而不经CPU,从而提升系统性能。然而,外设无法使用MMU将离散的物理内存区域转译为连续的虚拟内存区域,而外设通常需要连续的物理内存空间来进行DMA操作。SMMU则为外设提供所需的地址转译机制。当外设发起一个DMA请求,SMMU负责将离散的物理内存转换为连续的虚拟内存。该地址转译通常基于页表进行,称为SMMU页表,从而支持灵活的DMA操作。

[0035] 在SMMUv3中,每个外设被分配一个流ID,每个流ID对应于流表中的一个流表项(STE),流表项存储SMMU页表的基地址,以实现对多个外设的内存管理。

[0036] 基于以上对可信区架构及相关组件的介绍,描述本说明书中技术构思的实现。

[0037] 图1示出根据一个实施例的基于可信区架构创建的系统的结构示意图。在本说明书中,假设系统运行在支持Arm TrustZone可信区架构的平台上,并且所有异常级别都正常工作,不会发生越级访问。如图1所示,在正常世界中,运行着主机端,包括主机操作系统OS和其上运行的普通应用程序。根据本说明书实施例中的构思,为了给第三方应用提供可信执行环境,实施例的方案在正常世界中创建若干沙盒环境。所创建的沙盒是一个机密计算环境,可以保护在其中运行的应用程序(例如第三方部署的应用),并且可以在需要时独占访问外围设备。在一种实施方式下,实施例的方案能够生成多个并行的沙盒,这些沙盒与正常世界中的主机操作系统和应用程序完全隔离。

[0038] 在本说明书的各实施例方案中,假定安全世界的所有组件是完全可信的,主机端是不可信的。

[0039] 为了创建和管理上述沙盒环境,实施例的方案在TrustZone的原始安全监视器中添加了几个附加的软件组件,而无需进行任何硬件修改。具体的,添加的软件组件包括,内存保护器,SMMU管理器,沙盒管理器。

[0040] 内存保护器用于为沙盒提供隔离的内存访问。具体的,在实施例的方案中,通过为运行不同环境的不同CPU核提供不同的内存页表视图,强制沙盒与原始环境(即主机操作系统和应用程序)隔离执行。SMMU管理器通过将SMMU的管理转移到安全监视器,确保主机操作系统无法利用外设发起DMA攻击,以及确保不同环境对外设的独占访问权。沙盒管理器管理沙盒的整个生命周期(创建,运行,销毁),并通过安全监视器调用SMC提供接口,使第三方开发者可以快捷部署他们自己的沙盒来运行应用程序。

[0041] 下面描述基于图1的架构,为第三方应用提供可信执行环境的过程。图2示出根据一个实施例的基于可信区架构提供可信执行环境的方法。该方法基于图1所示的可信区架

构执行,其中可信区架构包括,安全世界和非安全世界;非安全世界运行有主机操作系统OS,安全世界的最高权限级(EL03)运行有安全监视器。该方法包括以下步骤。

[0042] 在步骤S21,响应于沙盒创建请求,非安全世界的主机操作系统OS为第一沙盒分配第一内存区段。

[0043] 在一个实施例中,可以在主机OS中额外部署两个内核模块CKM(Creat ion Kerna lModu le)和TKM(Terminat ion Kerna lModu le),CKM用于创建沙盒,TKM用于结束或销毁沙盒。当用户想要创建一个沙盒时,可以调用上述CKM发出沙盒创建请求,启动沙盒的创建过程。

[0044] 响应于用户的沙盒创建请求,主机OS(具体可以通过上述CKM)为该新创建的沙盒分配一个内存区段。由于实践中可以并行创建多个沙盒,区分起见,将这里新创建的沙盒称为第一沙盒,将为其分配的内存区段称为第一内存区段。

[0045] 然后,在步骤S22,运行于安全世界的安全监视器中的内存保护器,对运行主机OS的CPU核所使用的第一页表进行第一更新,使得更新后的第一页表中不包含对上述第一内存区段的映射。

[0046] 并且,在步骤S23,内存保护器创建第二页表,并将其设置为运行第一沙盒的CPU核使用的页表;其中第二页表中仅包含对第一内存区段的映射。

[0047] 通过步骤S22和S23可见,内存保护器为运行主机OS的CPU核,和运行第一沙盒的CPU核提供不同版本的页表视图,使得不同CPU核基于不同页表执行任务。本领域人员可以理解,页表用于提供虚拟地址到物理地址的映射。而在第一页表中,消除了第一内存区段的映射,这使得,使用第一页表的CPU核无法访问这一内存区段,即主机OS无法访问这一内存区段。而在第二页表中,仅包含第一内存区段的映射。这意味着,使用第二页表的CPU核仅可以访问第一内存区段,即,第一沙盒运行中仅可以访问第一内存区段。如此,通过为不同CPU核配置不同的页表视图,实现了内存访问的隔离。

[0048] 在以上方案中,将不同页表视图的管理放在安全监视器中,而不是在虚拟机监视器中执行这些操作。这是因为,虚拟机监视器会带来额外的负载并且攻击面较大。而通过安全监视器进行页表管理操作,可以将虚拟机监视器排除在可信基之外。

[0049] 在一个实施例中,上面提及的第一页表和第二页表,直接记录虚拟地址与内存物理地址的映射,即采用一级页表结构。不过更典型的,在多数Arm平台中,常采用二级页表或二阶段转译机制,来进行内存的访问控制。具体的,在二阶段转译机制中,内存的访问需要经过两个阶段的页表查询,即第一阶段S-1地址转译,和第二阶段S-2地址转译。在S-1地址转译中,内核控制一组转译表,将虚拟地址转译为中间物理地址IPA(I ntermed iate Phys ica l Address)。然后,通过虚拟机控制的一组页表,称为第二阶段页表S2PT(Stage-2Page Tab le),将IPA依次转译为真实的物理地址。

[0050] 在采用二阶段转译的情况下,步骤S22和S23中的第一页表和第二页表,均为第二阶段页表S2PT。

[0051] 更具体的,将第二页表设置为运行第一沙盒的CPU核使用的页表具体可以包括,将运行第一沙盒的CPU核的特定寄存器,设置为存储第二页表的基地址,如此,该CPU核就会将第二页表作为运行程序指令所依据的页表。在第二阶段页表的情况下,上述特定寄存器即VTTBR_EL2寄存器。

[0052] 在实现了内存隔离的基础上,在步骤S24,沙盒管理器启动第一沙盒,运行第三方部署在其中的目标应用,为目标应用提供可信执行环境。通过上述多版本页表提供的内存隔离,主机OS无法访问第一沙盒中运行的数据,实现沙盒与主机OS的隔离。此外,由于每个沙盒只能读取自己专属的内存区段,不同的沙盒之间也实现了隔离。从而,为目标应用提供了可信执行环境。

[0053] 根据一种实施方式,第一沙盒的运行需要使用外围设备。在Armv8平台架构下,假定各种外设在主机内存中具有内存映射MMIO区段。CPU通过上述MMIO区段使用对应的外围设备。而外设则基于前述SMMU页表对主机内存进行直接内存访问DMA。

[0054] 假定第一沙盒请求使用第一外设。此时,安全监视器(具体是内存保护器)在创建上述第二页表之后,还需对其进行更新,使得更新后的第二页表包含第一外设对应的内存映射MMIO区段中的映射。如此,第一沙盒可以通过该MMIO区段与第一外设交互,使用该第一外设。此外,安全监视器还通过SMMU管理器,对第一外设对应的SMMU页表进行更新,使得SMMU页表中仅包含上述第一内存区段的映射。这确保了第一外设仅可以访问第一沙盒的内存区段。

[0055] 在一种情况下,在第一沙盒请求使用第一外设之前或之时,第一外设原本是由主机OS使用的。此时,在主机OS完成针对第一外设的当前处理后(例如处理完第一外设的当前中断),安全监视器通过内存保护器对前述第一页表(即主机OS使用的页表)进行再次更新,该再次更新包括,清除第一页表中第一外设对应的MMIO区段中的映射。如此,主机OS无法继续访问该MMIO区段,也就无法继续使用第一外设。于是,第一外设从主机OS“解绑”。然后,通过前述更新第二页表和SMMU页表,将第一外设的使用权转移到上述第一沙盒。通过以上过程,确保外围设备在一个时间只被一个实体(主机OS或某个沙盒)独占使用。如此,也就防止了外围设备非法访问沙盒的内存,避免了外围设备的DMA攻击。

[0056] 图3示出在一个实施例中各个CPU核的内存页表视图。在图3的示例中,Arm平台具有4个CPU核,分别为核0,核1,核2以及核3。其中,在某个时刻,核0和核1中运行主机OS(及其上的普通应用),核2运行沙盒1,核3运行沙盒2。平台具有外设1和外设2。假定主机OS使用外设1,沙盒1使用外设2。

[0057] 整个物理内存空间可以划分为多个区段,具体包括:

[0058] SMMIO区段,用于SMMU内存映射;

[0059] PMMIO1区段,用于外设1的内存映射MMIO;

[0060] PMMIO2区段,用于外设2的内存映射MMIO;

[0061] NMem区段,普通世界的普通内存区段;

[0062] AMem1区段,分配给沙盒1的内存区段;

[0063] AMem2区段,分配给沙盒2的内存区段;

[0064] S2PT区段,用于存储第二阶段页表的区段;

[0065] SMem区段,安全世界的内存区段。

[0066] 各个CPU核均通过MMU页表进行内存访问。图3示出了各个CPU核的页表视图。

[0067] 核0和核1共同运行主机OS,共享同一页表视图,即第一行示出的页表视图,称为第一视图。在该第一视图中,仅PMMIO1和NMem是可以访问的,即,运行主机OS的CPU核可以通过PMMIO1与外设1交互,并可以基于普通内存区段NMem运行主机OS和其中的普通应用。由于沙

盒1使用了外设2,外设2对应的PMMIO2区段在该第一视图中不具有映射信息,无法访问。AMem1和AMem2分别分配给了沙盒1和沙盒2,这两个区段在该第一视图中也不具有映射信息,无法访问。其他区段属于安全世界才可以访问的区段。

[0068] 核2运行沙盒1,具有第二行示出的第二视图。在该第二视图中,仅包含PMMIO2和AMem1区段的映射信息,仅这2个区段是可以访问的。即,运行沙盒1的CPU核可以通过PMMIO2与外设2交互,并可以基于分配给沙盒1的专用内存区段AMem1运行沙盒1及其中的应用。该第二视图中,其他区段的映射信息均被清除或掩蔽,使得沙盒1无法访问其他区段。

[0069] 核3运行沙盒2,具有第三行示出的第三视图。在该第三视图中,仅包含AMem2区段的映射信息,即仅有这个区段是可以访问的。因此,沙盒2对应的CPU核仅可以基于分配给沙盒2的专用内存区段AMem2运行沙盒2及其中的应用。

[0070] 关于外设使用的SMMU页表,外设1和外设2也具有不同的页表视图。具体的,外设1被主机OS使用,因此在其SMMU页表视图中,仅主机OS对应的NMem区段是可以访问的,其他区段(沙盒1和沙盒2的区段)均因缺乏映射信息而无法访问。外设2被沙盒1使用,因此在其SMMU页表视图中,仅沙盒1对应的AMem1区段是可以访问的,其他区段(主机OS区段和沙盒2区段)均无法访问。

[0071] 从内存空间中各个区段的角度来说,用于外设1的PMMIO1区段仅可被主机OS访问,用于外设2的PMMIO2区段仅可被沙盒1访问,普通内存区段NMem可以被主机OS以及外设1访问,用于沙盒1的AMem1区段可以被沙盒1和外设2访问,用于沙盒2的AMem2仅可被沙盒2访问。图3中以不同的剪头形状,示出不同主体的访问。

[0072] 通过以上示例可以清晰看出,通过为不同CPU核提供不同页表视图,可以在不同运行主体(包括主机OS,各个沙盒)之间确保内存隔离。通过仅在单个实体的页表视图中体现外设MMIO区域的映射,使得每个外设在任意时刻只能被一个主体独占使用,并且,通过为不同外设配置SMMU页表,避免通过外设DMA对沙盒发起攻击。

[0073] 需要说明的是,尽管TZASC通过NSAID机制,可以实现针对外设群组的内存划分管理,但是,该机制存在一些局限。一方面,划分的各个区域必须是连续的,另一方面,划分的区域数量受限。这些限制和要求使得该机制难以适用于针对多个沙盒多个外设进行管理的场景。因此,根据上述实施例,在安全监视器中引入SMMU管理器,通过SMMU管理器对外设的SMMU页表进行配置和修改。通过该方式,可以灵活地针对沙盒环境配置外设的使用,避免来自外设的DMA攻击。

[0074] 下面结合一个具体示例,详细描述沙盒的完整生命周期。图4示出根据一个实施例的沙盒生命周期过程示意。

[0075] 首先是系统启动阶段。根据本说明书的实施例,可以在主机操作系统OS的启动代码中添加对页表进行初始化的代码。如此,在系统启动时,主机OS将会初始化其页表(图2中的第一页表)。典型的,该页表为第二阶段页表。随后,在安全监视器中,进行一系列SMMU初始化,这包括,初始化流表、命令队列等。然后,安全监视器通过SMMU管理器为每一个外设配置SMMU页表,即图4中所示的连接外设。以上操作都是主机操作系统通过调用对应的SMC接口完成的。

[0076] 系统启动完成后,用户可以通过调用创建沙盒的内核模块,即前述的CKM模块,来申请创建沙盒。系统进入创建沙盒的阶段。

[0077] 创建沙盒的阶段具体涉及以下步骤。首先,CKM请求内核(均属于主机OS)分配一段内存AMem,作为沙盒的内存空间,并返回内存的起始地址。接着,CKM将相关数据,包括内核镜像,内存文件系统,设备树等加载到该内存空间AMem中。然后CKM发起SMC调用,使CPU核进入EL3级,运行安全监视器SM。接着,安全监视器SM中的沙盒管理器验证数据的完整性后,为沙盒创建元数据,这些元数据包括,沙盒ID,物理地址空间,大小,上下文,以及针对沙盒的页表的基地址。此外,安全监视器中的内存保护器,从主机OS使用的页表(第一页表)中移除为沙盒分配的内存段AMem的映射关系。此外,内存保护器为沙盒创建一个新的页表(图2中的第二页表),其中仅包含内存段AMem的映射关系。

[0078] 在沙盒使用外设的情况下,内存保护器还需将该外设与主机OS解绑,并切换到沙盒。具体地,内存保护器从主机OS使用的页表中移除外设对应的MMIO区域,并在沙盒使用的页表中添加对应映射。此外,SMMU管理器从该外设的SMMU页表中移除主机OS的内存区段的映射,并添加沙盒的内存段AMem的映射,使得外设只能访问沙盒的内存。至此沙盒的创建完成,系统返回EL1,启动沙盒内核,进入沙盒运行阶段。

[0079] 根据一种实施方式,可以采用定制的Linux内核作为沙盒的操作系统。这样开发人员就可以任意使用Linux生态系统中的设备驱动程序,这些驱动程序可以在编译期间作为可加载内核模块(LKM)集成进来。

[0080] 相应的,在沙盒的运行阶段,可以从上述定制的Linux开始启动,沙盒使用的外设驱动已经作为内核模块和Linux编译在一起了,这样的设计可以让开发者能够便捷地使用到当前Linux生态系统中的所有外设驱动。

[0081] 在沙盒中的应用运行完成后,会通过SMC接口再次进入安全监视器(具体可以是,主机OS内核中的TKM向安全监视器发送SMC),进入结束沙盒的阶段。

[0082] 在结束/销毁沙盒的阶段,安全监视器中的沙盒管理器会清除沙盒的内存和缓存,销毁其元数据。随后返回主机操作系统OS中的内核模块,内核模块会释放之前分配的内存,至此,一个沙盒的生命周期就结束了。

[0083] 回顾以上过程,在以上实施例的方案中,取代于虚拟化的方式,只是在正常世界中创建轻量级的若干沙盒,用于部署第三方应用。通过安全监视器为运行不同环境的不同CPU核提供不同的内存页表视图,强制沙盒与主机操作系统,以及沙盒与沙盒之间保持内存隔离。SMMU管理器通过将SMMU的管理转移到安全监视器,确保主机操作系统无法利用外设发起DMA攻击,以及确保不同环境对外设的独占访问权。如此,以轻量级的沙盒环境的方式,为第三方应用提供可信执行环境。

[0084] 另一方面,与上述方法过程相对应的,本说明书实施例还披露一种基于可信区架构提供可信执行环境的装置,所述可信区架构包括,安全世界和非安全世界;所述装置包括,运行于非安全世界的主机操作系统OS,运行于安全世界的安全监视器,所述安全监视器包括内存保护器,沙盒管理器,其中:

[0085] 所述主机OS配置为,响应于沙盒创建请求,为第一沙盒分配第一内存区段;

[0086] 所述内存保护器配置为,对运行主机OS的CPU核所使用的第一页表进行第一更新,使得更新后的第一页表中不包含对所述第一内存区段的映射;

[0087] 所述内存保护器还配置为,创建第二页表,并将其设置为运行所述第一沙盒的CPU核使用的页表;其中所述第二页表中仅包含对所述第一内存区段的映射;

[0088] 所述沙盒管理器配置为,启动所述第一沙盒,运行第三方部署在其中的目标应用,为目标应用提供可信执行环境。

[0089] 根据一种实施方式,所述安全监视器还包括SMMU管理器;

[0090] 所述内存保护器还配置为,响应于第一沙盒请求使用第一外设,更新第二页表,使得更新后的第二页表包含所述第一外设对应的内存映射MMIO区段中的映射;

[0091] 所述SMMU管理器配置为,对所述第一外设对应的SMMU页表进行更新,使得SMMU页表中仅包含所述第一内存区段的映射。

[0092] 进一步的,在一个实施例中,在所述第一沙盒请求使用第一外设之前或之时,所述第一外设由所述主机OS使用;相应的:

[0093] 所述内存管理器还配置为,对所述第一页表进行第二更新,所述第二更新包括,清除所述第一页表中所述第一外设对应的MMIO区段中的映射;

[0094] 所述SMMU管理器具体配置为,从所述SMMU页表中移除所述主机OS使用的内存段的映射,并添加所述第一内存区段的映射。

[0095] 上述装置的具体执行过程示例,可以参照之前结合图1和图2的描述,不复赘述。

[0096] 根据再一方面的实施例,还提供一种计算设备,包括存储器和若干处理器,所述存储器中存储有可执行代码,所述处理器执行所述可执行代码时,实现上述结合图2描述的方法。

[0097] 本领域技术人员应该可以意识到,在上述一个或多个示例中,本发明所描述的功能可以用硬件、软件、固件或它们的任意组合来实现。当使用软件实现时,可以将这些功能存储在计算机可读介质中或者作为计算机可读介质上的一个或多个指令或代码进行传输。

[0098] 以上所述的具体实施方式,对本发明的目的、技术方案和有益效果进行了进一步详细说明,所应理解的是,以上所述仅为本发明的具体实施方式而已,并不用于限定本发明的保护范围,凡在本发明的技术方案的基础之上,所做的任何修改、等同替换、改进等,均应包括在本发明的保护范围之内。

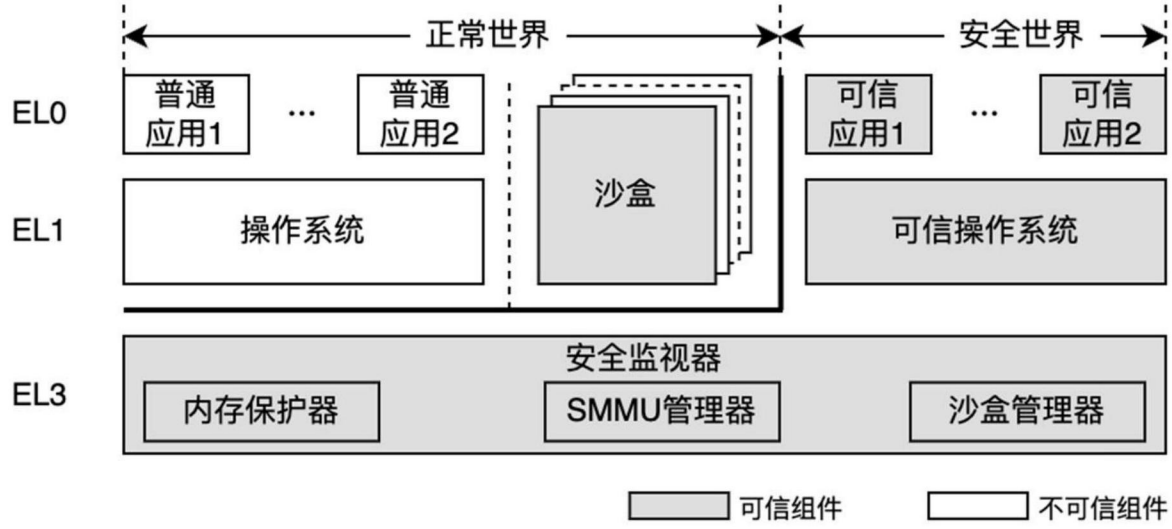

正常世界
安全世界
EL0
普通应用1
普通应用2
沙盒
可信应用1
可信应用2
EL1
操作系统
可信操作系统
EL3
安全监视器
内存保护器
SMMU管理器
沙盒管理器
可信组件
不可信组件

图1

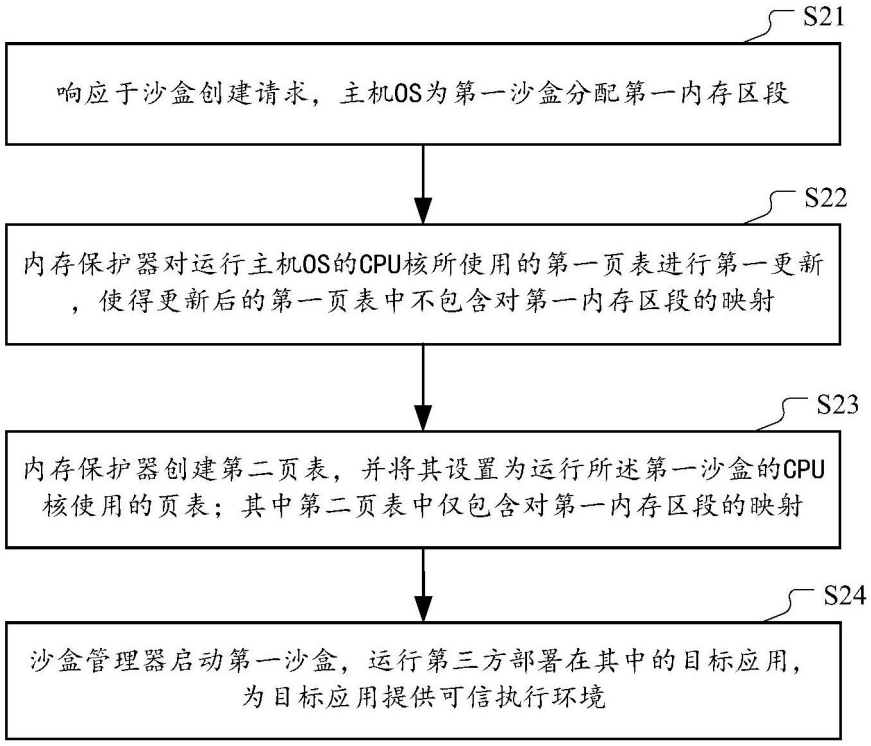

S21
响应于沙盒创建请求，主机OS为第一沙盒分配第一内存区段
S22
内存保护器对运行主机OS的CPU核所使用的第一页表进行第一更新，使得更新后的第一页表中不包含对第一内存区段的映射
S23
内存保护器创建第二页表，并将其设置为运行所述第一沙盒的CPU核使用的页表；其中第二页表中仅包含对第一内存区段的映射
S24
沙盒管理器启动第一沙盒，运行第三方部署在其中的目标应用，为目标应用提供可信执行环境

图2

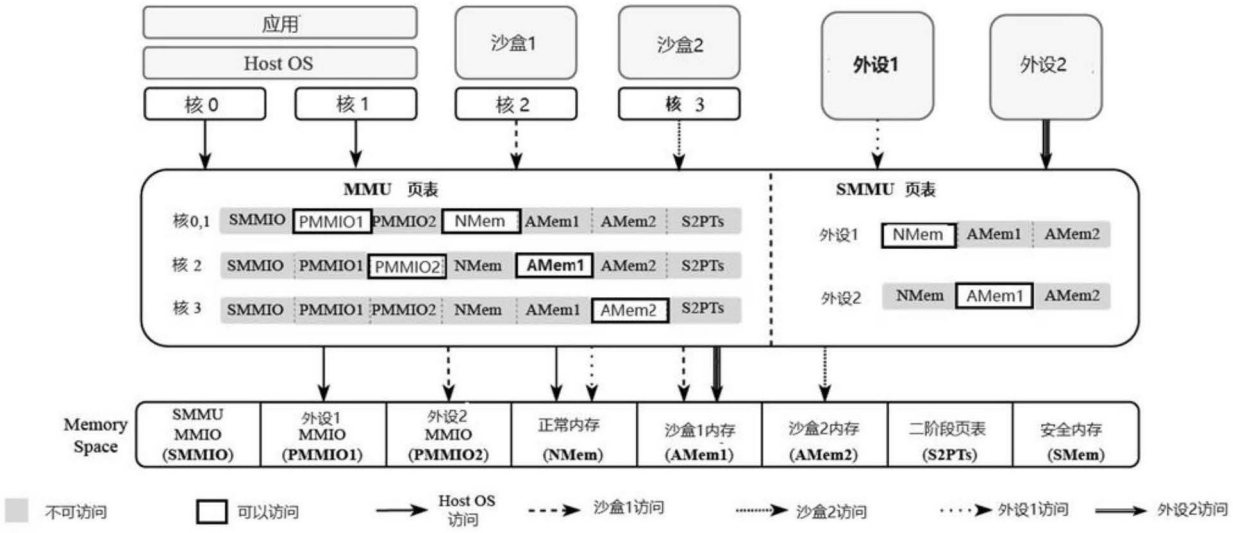

应用
Host OS
核 0
核 1
沙盒1
核 2
沙盒2
核 3
外设1
外设2
MMU 页表
核0,1 SMMIO PMMIO1 PMMIO2 NMem AMem1 AMem2 S2PTs
核 2 SMMIO PMMIO1 PMMIO2 NMem AMem1 AMem2 S2PTs
核 3 SMMIO PMMIO1 PMMIO2 NMem AMem1 AMem2 S2PTs
SMMU 页表
外设1 NMem AMem1 AMem2
外设2 NMem AMem1 AMem2
Memory Space
SMMU MMIO (SMMIO)
外设1 MMIO (PMMIO1)
外设2 MMIO (PMMIO2)
正常内存 (NMem)
沙盒1内存 (AMem1)
沙盒2内存 (AMem2)
二阶段页表 (S2PTs)
安全内存 (SMem)
不可访问
可以访问
Host OS 访问
沙盒1访问
沙盒2访问
外设1访问
外设2访问

图3

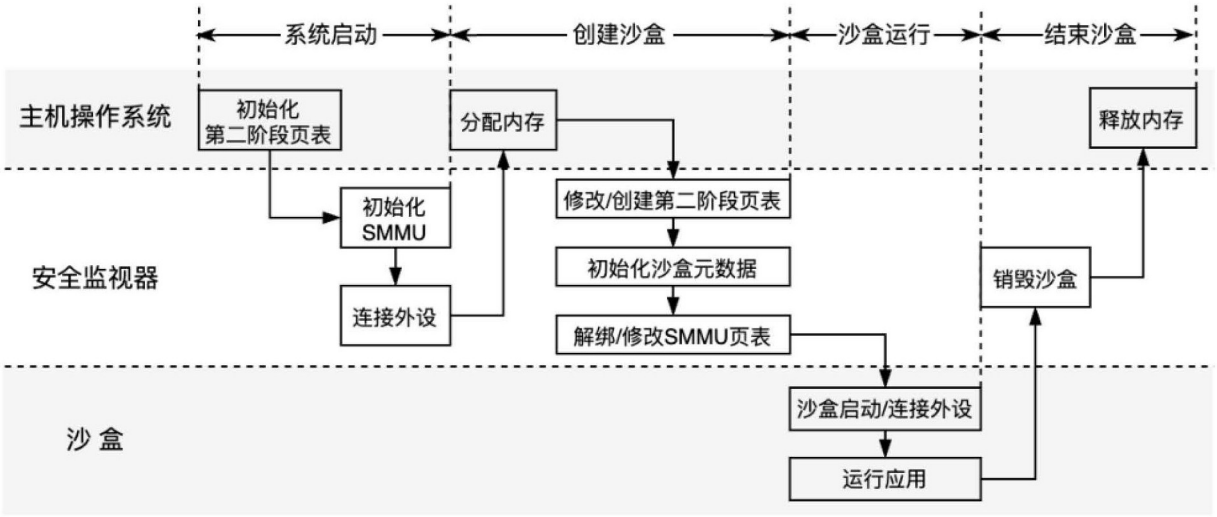

系统启动
创建沙盒
沙盒运行
结束沙盒
主机操作系统
初始化 第二阶段页表
分配内存
释放内存
初始化 SMMU
修改/创建第二阶段页表
安全监视器
连接外设
初始化沙盒元数据
销毁沙盒
解绑/修改SMMU页表
沙盒启动/连接外设
沙 盒
运行应用

图4