软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*, 2022,33(6):2189−2207 [doi: 10.13328/j.cnki.jos.006577]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

# 基于精化的可信执行环境内存隔离机制验证*

靳翠珍[1], 张倩颖[1,2,3], 马雨薇[1], 李希萌[1,4], 王国辉[1,2], 施智平[1,4], 关 永[1,5]

[1](首都师范大学 信息工程学院, 北京 100048)

[2](高可靠嵌入式系统北京市工程研究中心(首都师范大学), 北京 100048)

[3](计算机体系结构国家重点实验室(中国科学院 计算技术研究所), 北京 100190)

[4](电子系统可靠性技术北京市重点实验室(首都师范大学), 北京 100048)

[5](电子系统可靠性与数理交叉学科国家国际科技合作示范型基地(首都师范大学), 北京 100048)

通信作者: 张倩颖, E-mail: qyzhang@cnu.edu.cn

**摘 要**: 可信执行环境(trusted execution environment, TEE)基于硬件隔离机制, 为安全敏感应用提供隔离的执行环境, 保护敏感数据的安全性. 内存隔离机制是 TEE 的关键机制之一, 用于对安全内存和非安全内存进行隔离, 并对安全内存实施访问控制, 如果其安全性不能保证, 可能造成存储在安全内存中的敏感数据泄露. 为验证 TEE 内存隔离机制的安全性, 针对基于 ARM TrustZone 技术构建的 TEE, 提出一种基于精化的可信执行环境内存隔离机制安全性验证方法. 建立抽象模型和具体模型, 并定义两种模型之间的精化关系, 在证明精化关系成立和抽象模型满足信息流安全性的前提下, 验证具体模型的信息流安全性. 具体模型建模了 TEE 内存隔离机制的关键硬件和软件, 包括 TrustZone 地址空间控制器、MMU 和 TrustZone monitor 等, 在定理证明器 Isabelle/HOL 中, 验证了该模型满足无干扰、无泄露、无影响等信息流安全属性.

**关键词**: TEE; 内存隔离机制; 形式化验证; 精化关系; 信息流安全性

**中图法分类号**: TP311

中文引用格式: 靳翠珍, 张倩颖, 马雨薇, 李希萌, 王国辉, 施智平, 关永. 基于精化的可信执行环境内存隔离机制验证. 软件学报, 2022, 33(6): 2189–2207. http://www.jos.org.cn/1000-9825/6577.htm

英文引用格式: Jin CZ, Zhang QY, Ma YW, Li XM, Wang GH, Shi ZP, Guan Y. Refinement-based Verification of Memory Isolation Mechanism for Trusted Execution Environment. Ruan Jian Xue Bao/Journal of Software, 2022, 33(6): 2189–2207 (in Chinese). http://www.jos.org.cn/1000-9825/6577.htm

## Refinement-based Verification of Memory Isolation Mechanism for Trusted Execution Environment

JIN Cui-Zhen[1], ZHANG Qian-Ying[1,2,3], MA Yu-Wei[1], LI Xi-Meng[1,4], WANG Guo-Hui[1,2], SHI Zhi-Ping[1,4], GUAN Yong[1,5]

[1](College of Information Engineering, Capital Normal University, Beijing 100048, China)

[2](Beijing Engineering Research Center of High Reliable Embedded System (Capital Normal University), Beijing 100048, China)

[3](State Key Laboratory of Computer Architecture (Institute of Computing Technology, Chinese Academy of Sciences), Beijing 100190, China)

[4](Beijing Key Laboratory of Electronic System Reliability Technology (Capital Normal University), Beijing 100048, China)

* 基金项目: 国家自然科学基金(61802375, 61602325, 61876111, 61877040, 62002246); 北京市教委科技计划(KM201910028005, KM202010028010); 中国科学院计算技术研究所计算机体系结构国家重点实验室开放课题(CARCH201920); 中央支持地方建设-“双一流”建设项目(20531120005)

本文由“定理证明理论与应用”专题特约编辑曹钦翔副教授、詹博华副研究员、赵永望教授推荐.

收稿时间: 2021-09-04; 修改时间: 2021-10-14; 采用时间: 2022-01-04; jos 在线出版时间: 2022-01-28

[5](International Science and Technology Cooperation Base of Electronic System Reliability and Mathematical Interdisciplinary (Capital Normal University), Beijing 100048, China)

**Abstract**: A trusted execution environment (TEE) provides security-sensitive applications with an isolated execution environment based on hardware isolation mechanisms to protect sensitive data. The memory isolation mechanism is one of the key mechanisms of TEE, which is used to isolate the secure memory from the non-secure memory and to control the access to the secure memory. If the security of the memory isolation mechanism can not be guaranteed, sensitive data stored in the secure memory may be leaked. To verify the security of the TEE memory isolation mechanism, a refinement-based security verification method of the memory isolation mechanism is propose for ARM TrustZone-based TEE. An abstract model and a concrete model are established, a refinement relation between these two models is defined, and the information flow security of the concrete model is verified on the premise of proving that the refinement relation is held and the abstract model satisfies the information flow security properties. The proposed concrete model consists of key hardware and software of the TEE memory isolation mechanism, including TrustZone address space controller, MMU, and TrustZone monitor, and using the theorem prover Isabelle/HOL, it is verified that the concrete model satisfies the information flow security properties, such as noninterference, nonleakage, and noninfluence.

**Key words**: TEE; memory isolation mechanism; formal verification; refinement relation; information flow security

智能手机、平板电脑等嵌入式设备广泛应用于生产生活的各个领域, 其上运行的软件和存储的敏感数据随之增加. 然而, 操作系统、第三方软件等特权软件和应用程序的漏洞为攻击者提供了可乘之机. 随着运行的软件数量的增加, 嵌入式设备被攻击的风险也随之增加, 一旦受到攻击, 可能会造成敏感数据泄露或被破坏, 给用户个人隐私或财产安全造成损害. 为保证嵌入式设备中敏感数据的安全性, 可信执行环境(trusted execution environment, TEE)的概念[1]被提出. TEE 利用硬件机制, 为敏感数据和可信应用程序提供一个与其他执行环境隔离的安全区域, 保证敏感数据不会被泄露. 之后, TEE 的概念被加入到硬件设计中. 目前, 支持实现 TEE 的主流硬件技术有 ARM TrustZone[2]、Intel SGX[3]和 AMD SEV[4]等.

ARM TrustZone 技术自提出后, 得到移动和嵌入式设备厂商的支持, 被广泛应用于数字版权保护、移动金融服务和认证等领域. 本文即针对基于 ARM TrustZone 技术构建的 TEE 展开研究, 如无特殊说明, 后文中的 TEE 均指基于 ARM TrustZone 技术的 TEE 架构.

ARM TrustZone 技术对片上系统进行安全扩展, 为处理器提供安全世界和普通世界两种执行模式. 其中, 安全世界用于运行 TEE, 普通世界用于运行通用执行环境(rich execution environment, REE). 处理器通过监视器(monitor)模式在两者之间进行切换. TrustZone 内存隔离机制将内存划分为安全内存和非安全内存, 分别分配给安全世界和普通世界, 并对内存访问实施控制, 以便在两个世界间实施单向隔离: 当处理器运行在安全世界时, 可以访问安全内存和非安全内存; 当处理器运行在普通世界时, 只能访问非安全内存, 不能访问安全内存. 这种单向隔离保证存储在安全内存中的敏感数据不会泄露到普通世界. 内存隔离机制是 TEE 的关键机制, 其安全性对 TEE 的安全性至关重要. 如果不能保证内存隔离机制的安全性, 会导致 REE 非法读取 TEE 内存, 破坏 TEE 敏感数据的安全性. 因此, 对内存隔离机制进行形式化验证是必要的.

目前, 关于内存隔离机制的形式化验证工作[5−10]存在两方面的问题.

- 首先, 针对可信执行环境内存隔离机制的验证工作较少且已有的工作未完整建模内存隔离机制组件. 文献[5,6]验证了 hypervisor 内存隔离机制的信息流安全性; 文献[7]虽然验证了可信执行环境内存隔离机制的信息流安全性, 但其缺少对 MMU 等关键性组件的建模. 可信执行环境是涉及硬件和软件两方面的复杂结构, 因此, 要验证其内存隔离机制的安全性不仅要考虑软件, 还需要考虑硬件组件;
- 其次, 现有工作验证的内存隔离属性仅仅是内存隔离的正确性和简单的安全属性. 文献[8,9]验证了内核中内存隔离机制的正确性; 文献[6,10]验证了内核中内存隔离机制的机密性、完整性和一致性等简单安全属性. 然而, 内存隔离涉及隔离域间的信息流问题, 为保证其安全性, 应对内存隔离机制的信息流安全性等复杂安全属性进行验证.

针对上述问题, 本文提出一种基于精化的 TEE 内存隔离机制安全性验证方法, 能够对包含硬件和软件在内的内存隔离机制进行细粒度的建模, 并对其信息流安全属性[11]进行验证. 本文依据 ARM 规范[12]并参考

ARM 可信固件(ARM trusted firmware, ATF)[13]建模 TEE 内存隔离机制. 首先提出一个新的抽象模型, 建模 TrustZone 中与内存隔离相关的硬件和软件组件, 包括处理器、内存、TrustZone 地址空间控制器(TrustZone address space controller, TZASC)和 TrustZone 监视器(monitor), 并建模安全性验证涉及的安全域和敌手; 然后对抽象模型进行精化建立具体模型, 其中增加了新组件 MMU, 并将其功能建模为 mapping 事件; 之后, 为保证模型中组件功能的正确性以及模型与实际 TEE 内存隔离机制的一致性, 本文证明了两个模型的正确性属性. 在此基础上, 本文首先分析 TrustZone 访问控制模型, 提出两种安全策略; 然后定义内存隔离机制的无干扰、无泄露和无影响等信息流安全属性; 之后, 在证明模型间的精化关系和抽象模型的信息流安全性的情况下, 验证了具体模型在两种安全策略下均满足信息流安全性.本文的建模和验证均在定理证明器 Isabelle/HOL 中进行, 且在验证过程中发现: 若要保证 TEE 的机密性, 需要限制以下操作: 安全世界向普通世界的内存写、普通世界内存区域的使能和普通世界内存区域的禁用.

本文的创新工作如下:

(1) 提出了 TEE 内存隔离机制的抽象模型. 在文献[7]的基础上, 提出一个新的抽象模型: 更加完整的建模 TrustZone 监视器的功能, 其在世界切换时可以对上下文进行保存和恢复; 根据 TrustZone 的安全模型, 在抽象模型中划分出 TEE、REE 和 MON 这 3 个安全域, 以支持对安全策略的细粒度描述; 建模敌手能力, 允许定义不同能力的敌手, 以支持不同敌手模型下的安全属性验证;

(2) 精化了抽象模型建立内存隔离机制的具体模型. 具体模型中, 增加了内存管理单元 MMU 组件的模型. 该模型包含遍历转换表和查询 TLB 两种页表转换方式, 并将 MMU 组件功能建模为 mapping 事件. 为保证对 MMU 建模的正确性, 分别验证了遍历转换表和查询 TLB 这两种页表转换功能的正确性. 具体模型完整建模了虚拟地址到物理地址的转换过程, 是对 TEE 内存隔离机制更细粒度的建模;

(3) 提出了内存隔离机制的两种安全策略. 通过分析 TrustZone 内存访问控制模型, 提出了两种信息流安全策略: 不允许 TEE 向 REE 有直接信息流; 允许 TEE 向 REE 有直接信息流. 上述安全策略分别对应 TrustZone 的两种应用场景, 其中, 第 1 种安全策略对应需要保护 TEE 中数据机密性的应用场景. 本文在这两种安全策略下, 验证 TEE 内存隔离机制满足无干扰、无泄露和无影响等信息流安全属性;

(4) 发现了破坏 TEE 中数据机密性的 3 种内存操作. 分析 TEE 内存隔离机制建模和验证过程以及验证结果发现: 不仅安全世界写非安全内存的操作会引起TEE到REE的信息流, 普通世界内存区域的使能和禁用两个操作也会引起TEE到REE的信息流. 在第1种安全策略下, 若不对上述操作进行限制, 则会造成系统中隐蔽的信息流通道. 为保护TEE中敏感数据的机密性, 高安全要求的TEE用户应对上述 3 个操作施加限制.

本文第 1 节简介与本文研究内容相关的背景知识. 第 2 节介绍 TEE 内存隔离机制的抽象模型及其正确性证明. 第 3 节描述在抽象模型基础上精化得到的具体模型及其正确性证明. 第 4 节阐述具体模型的安全性验证过程, 包括抽象模型的信息流安全性验证、精化关系的证明以及具体模型的信息流安全性验证. 第 5 节介绍本文的相关研究工作. 第 6 节总结全文.

## 1 技术背景

本节给出与本文相关的背景知识. 首先介绍 ARM TrustZone 技术, 重点介绍内存隔离机制中的地址转换和地址空间隔离; 其次简介基于精化的验证方法; 然后简述信息流安全性证明方法; 最后介绍本文使用的验证工具定理证明器 Isabelle/HOL 及其相关的语法.

### 1.1 ARM TrustZone技术

ARM TrustZone 架构[14]为处理器提供安全世界和普通世界两种执行模式: 安全世界用于安全子系统, 运行可信操作系统和可信应用程序; 普通世界用于非安全子系统, 运行通用操作系统和不可信应用程序,

TrustZone 通过硬件隔离，保证普通世界不能访问安全世界的资源. 两个世界通过在 monitor 模式修改安全配置寄存器(secure configuration register, SCR)中 *NS* 位的值进行切换：当 *NS* 位的值为 0 时，处理器运行在安全世界中；当 *NS* 位的值为 1 时，处理器运行在普通世界中. 两个世界进行切换时，monitor 保存处理器切换前的世界状态，并恢复处理器切换后的世界状态，需要保存和恢复的世界状态包括通用寄存器、系统寄存器以及包含浮点寄存器在内的其他寄存器.

TrustZone 内存隔离机制将内存划分为安全内存和非安全内存，保证安全内存只能被安全世界访问，不能被普通世界访问；而非安全内存则可以被安全世界和普通世界访问. TrustZone 的内存隔离是由 MMU 组件和 TZASC 组件共同保证的，其整体架构如图 1 所示. MMU 是处理器用来管理虚拟存储器和物理存储器的控制组件，负责控制内存访问权限和将虚拟地址映射为物理地址. TZASC 是高性能的地址空间控制器，可以对内存进行划分并对内存访问实施控制.

MMU 是内存隔离的一个重要组件，有控制内存访问权限和地址转换两种功能[15]. 内存访问有 3 种权限，分别是可读可写、只读和不能访问. 内存访问权限由表描述符中 *APTable* 位的值和页描述符中 *AP* 位的值来决定: *APTable* 位的值指定下一级转换表的访问权限, *AP* 位的值指定物理内存的访问权限. 虚拟地址到物理地址转换时, MMU 首先检查 TLB 是否命中：如果命中，则地址转换可立即执行；若未命中, MMU 通过遍历转换表进行地址转换. 在地址转换中，非安全的虚拟地址只能被映射到非安全的物理地址，而安全的虚拟地址则根据表描述符中 *NSTable* 位的值和页描述符中 *NS* 位的值来决定映射到安全的物理地址还是非安全的物理地址.

- 当前转换表位于安全内存时，*NSTable* 位的值为 0，则下一级转换表位于安全内存中；*NSTable* 位的值为 1，则下一级转换表位于非安全内存中；
- 当前转换表位于安全内存时，*NS* 位的值为 0，则映射到安全物理地址；*NS* 位的值为 1，则映射到非安全物理地址.

MMU 保证安全内存不会被非安全的虚拟地址映射到，为内存隔离机制提供了保障.

TZASC 用于划分内存和控制内存访问. TZASC 可将内存划分为 2、4、8 或 16 个区域，每个区域又可以划分为 8 个大小相等且互不重叠的子区域. 每个区域有安全权限和区域使能等参数：通过编写安全权限参数，可以将内存区域划分为安全内存区域或非安全内存区域；安全世界通过内存区域使能和禁用操作设置区域使能参数，其值为 0 时，区域被禁用；其值为 1 时，区域被使能. TZASC 通过比较内存区域安全权限参数的值和请求访问内存区域时代表处理器所处世界的 *NS* 位的值来对内存访问进行控制，拒绝非安全事务对安全内存的访问.

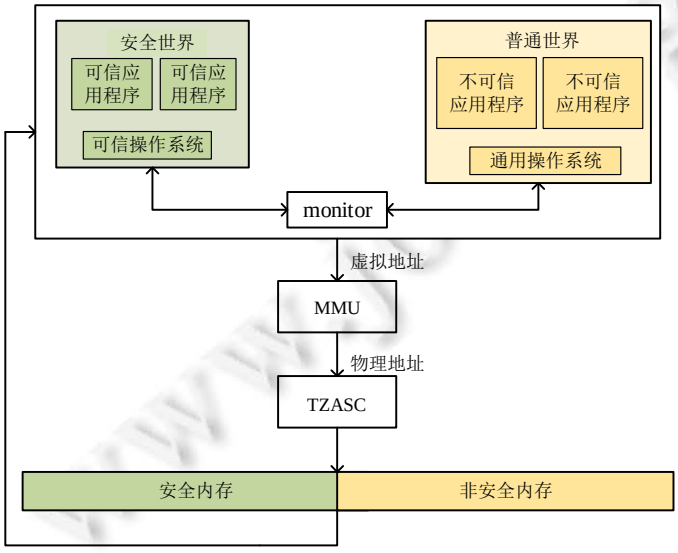


图 1　ARM TrustZone 架构的内存隔离机制

### 1.2 精化方法

精化是一种程序构造技术[16], 其思想是: 从抽象规范开始, 通过添加细节或消除不确定性等一系列精化步骤, 来逐步开发程序. 之后, 精化方法作为一种通用技术被用于形式化验证领域, 其思想是: 通过对一个抽象模型进行逐步证明精化, 构造一个具体模型的证明过程.

精化证明通过在抽象模型和具体模型之间建立精化关系来验证精化的正确性, 精化关系 $R$ 将抽象和具体模型的状态对应起来. 精化关系可以根据程序规范(如精化演算[17])或模拟关系来表达和证明. 精化演算通过使用精化定律(refinement law)从前一个模型计算每个中间模型, 这些定律的应用能够减少证明义务并确保精化的正确性. 模拟关系[18]是两个模型状态间的对应状态, 模拟证明建立在这种对应关系的基础上, 两个系统之间存在模拟, 表明一个系统的任何行为也可以由另一个系统展现出来(模拟). 以前向模拟为例, 如果对于具体模型 $M_c$ 中的每个从初始状态 $s$ 到状态 $s'$的迁移, 在抽象模型 $M_a$ 中都存在一个从状态$\sigma$到$\sigma'$的相应迁移, 则称 $M_c$ 精化 $M_a$. 所谓相应的迁移是指: 如果状态 $s$ 和$\sigma$之间存在关系 $R$, 对于状态 $s'$和抽象状态$\sigma'$, $R$ 在它们之间仍成立. 即: 在两个模型的迁移过程中, 得到的状态之间都满足关系 $R$.

精化建立了抽象模型与具体模型之间的对应关系, 表明对抽象模型的分析结果也可以转移到具体模型中. 由于在抽象模型上验证属性通常更容易, 基于精化的验证方法可以简化验证过程.

### 1.3 信息流安全性证明方法

系统可以被划分为不同的域, 域是一个抽象的概念, 表示操作的权限. 信息流策略规定了信息在不同的域之间所允许的流动方向. 信息流安全性用来保证系统中不存在违反安全策略的隐蔽信息流通道. 目前有两种主要的信息流安全属性: 无干扰信息流和 GWV(以其作者 Greve, Wilding 和 Vanfleet 的名字命名)策略[19]. 本文使用无干扰信息流, 无干扰信息流有 3 个子属性, 分别为无干扰、无泄露和无影响.

- 无干扰属性是指在安全策略中不允许向观察域有信息流动的事件的发生不改变观察结果. 即: 对于一个不允许向观察域有信息流动的事件, 观察者不能分辨出执行和不执行此事件的区别. 为定义无干扰属性, 引入了清除函数 *ipurge*, 该函数将在安全策略中不允许向域 $d$ 有信息流动的事件清除. 无干扰属性定义为: 某一状态执行原事件序列后得到的状态和执行进行清除操作后的事件序列得到的状态关于域 $d$ 等价;
- 无泄露属性确保如果机密数据在最初没有被泄露, 那么在系统执行期间也不会被泄露. 即, 当前域的信息不会因事件执行而被外部非法获取. 为定义无泄露属性, 引入了函数 *sources*, 该函数根据安全策略返回执行某事件序列时允许向域 $d$ 有信息流的域集合. 无泄露属性定义为: 如果两个状态关于域 $d$ 等价且关于 *sources* 函数返回的域集合中的域也等价, 那么这两个状态分别执行该事件序列后得到的状态关于域 $d$ 仍等价;
- 无影响属性是无干扰属性和无泄露属性的结合, 确保不允许向域 $d$ 有信息流动的事件不会对 $d$ 的观察造成影响以及 $d$ 的机密数据不会泄露. 无影响属性定义为: 两个状态关于域 $d$ 等价且关于允许向域 $d$ 有信息流的域也等价, 那么这两个状态分别执行进行清除操作后的事件序列(原事件序列相同)得到的状态关于域 $d$ 仍等价. 上述属性的具体定义详见文献[20].

信息流安全属性的证明通常采用展开定理(unwinding theorem), 通过证明执行轨迹中各个执行步骤都满足展开条件, 实现对整个系统满足安全属性的证明. 本文通过证明信息流安全属性推理框架[21]可知: 无影响属性可推导出无干扰属性和无泄露属性, 而无影响属性则可从展开条件推导得出. 展开条件为 *local_respect* 和 *weak_step_consistent*, 其中,

- *local_respect* 定义为: 若事件的执行域不允许有流向观察域的信息流, 则当前状态与其执行该事件后得到的新状态关于观察域是等价的;
- *weak_step_consistent* 定义为: 若两个状态关于观察域和事件的执行域都是等价的, 则分别执行相同事件后得到的新状态关于观察域仍是等价的.

### 1.4 Isabelle/HOL定理证明器

定理证明器Isabelle/HOL由函数式编程语言ML实现，支持对系统进行形式化建模，以及用证明策略或结构化证明语言验证其属性[22]. 目前, Isabelle/HOL已被广泛应用于数学公式或计算机系统等方面的形式化验证工作.

在Isabelle/HOL中，关键字 *datatype* 用于定义数据类型. 常用的类型定义如下:

*datatype 'a option=None|Some 'a.*

其表示数据类型*'a option* 是在数据类型*'a* 的基础上增加一个在通用意义上代表空的元素 *None*. 此外，关键字 *type_synonym* 用于对一些复杂类型定义进行缩写. 普通函数通常使用关键字 *definition* 定义，递归函数则使用关键字 *primrec* 定义.

Isabelle/HOL 中的结构化证明语言称为“Isar”. 结构化证明是正向证明，即从已知事实出发逐步推导出待证明的结论，其语法格式为 *proof*[*method*]，表示：应用初始证明方法 *method*，并进入结构化证明状态. 若 *method* 未给出，将根据目标命题的形式自动选择初始证明方法；可使用“-”(连字符)作为初始证明方法，“-”可理解为不使用任何证明方法的空方法.

## 2 内存隔离机制的抽象模型

本节建立基于ARMv8 TrustZone架构的TEE内存隔离机制的抽象模型，该抽象模型包括内存模型和敌手模型. 内存模型建模内存和关键寄存器的状态，并且对与内存隔离机制相关的组件TZASC和monitor的功能进行抽象描述，包括内存划分、内存区域使能控制、访问控制和世界切换等. 另外，基于抽象状态建模内存读写等事件. 敌手模型建模敌手可观察到的系统状态和敌手的能力. 下面首先对抽象模型进行概述，然后分别介绍内存模型、敌手模型和抽象模型的正确性定义及证明.

### 2.1 抽象模型概述

本节在抽象层次上建立内存隔离机制的新抽象模型，该模型以文献[7]中的内存模型为基础模型，针对精化验证需求对其进行了相应的修改：重新划分安全域；增加 monitor 组件执行世界切换时保存上下文的功能；增加敌手模型. 本节介绍新模型重新划分的安全域，另外两个新增加的内容分别在第 2.2 节和第 2.3 节介绍.

根据TrustZone的安全模型和信息流情况，本文将内存模型划分为3个不同的域: TEE、REE和MON，以支持对信息流策略的细粒度描述. 首先，安全世界和普通世界具有不同的安全权限，应将其分别划分为 TEE 和 REE 两个安全域. 然后，分析两个世界间的信息流可知，世界切换时保存上下文的安全内存和普通安全内存相比有更复杂的信息流：安全世界切换到普通世界时，在恢复普通世界上下文过程中产生TEE到REE的直接信息流；普通世界切换到安全世界时，在保存普通世界的上下文过程中产生REE到TEE的信息流. 为精确描述上述信息流并制定细粒度安全策略，本文将保存上下文的安全内存划分至安全域MON，该域不能被除世界切换之外的其他事件影响.

TEE 内存隔离机制的抽象模型是一个五元组，由内存模型、TrustZone 安全域、TrustZone 安全策略、等价关系和敌手模型组成，该抽象模型 $TEE\_M_a$ 的定义如下.

**定义 1(抽象模型)**:

$$TEE\_M_a=(M_a,D,\rightsquigarrow,\sim,A_a),$$

其中,

(1) $M_a$ 是内存模型，该模型抽象描述了与内存隔离机制相关的硬件组件和软件组件. 其中，硬件组件有处理器、内存和TZASC，软件组件有monitor. 将组件TZASC和monitor的功能建模为事件: TZASC组件的内存区域使能控制功能建模为内存区域使能事件和内存区域禁用事件、monitor 组件的功能建模为世界切换事件;

(2) $D$={TEE,REE,MON}是安全域集合，其中, TEE表示安全世界域, REE表示普通世界域, MON表示世

界切换时保存上下文的域;

(3) $\rightsquigarrow \subseteq D\times D$ 表示 TrustZone 的安全策略. 假定 $d_1$ 和 $d_2$ 表示 $D$ 中两个不同的域, 则 $d_1 \rightsquigarrow d_2$ 表示允许域 $d_1$ 有信息流流向域 $d_2$; $d_1 \not\rightsquigarrow d_2$ 表示不允许域 $d_1$ 有信息流流向域 $d_2$;

(4) ~是等价关系, 包括状态等价和观察等价, 具体定义见第 4.1.2 节;

(5) $A_a$ 是敌手模型, 本文建模的敌手是在普通世界具有内核级权限的敌手, 可对普通世界的内存进行读写操作. 本文的敌手模型支持扩展, 可根据不同的威胁模型扩展该模型, 以建模不同能力的敌手.

下面分别对内存模型和敌手模型进行详细介绍.

### 2.2 内存模型

内存模型建模内存隔离机制的内存和寄存器状态, 对相关组件 TZASC, monitor 进行抽象描述, 基于抽象状态将组件功能建模为事件: 内存区域使能 *region_enable* 和内存区域禁用 *region_disable*、世界切换 *world_switch*、内存读 *mem_read* 和内存写 *mem_write*.

如前文所述, 安全世界的内存划分为两部分, 分别属于安全域 TEE 和安全域 MON, 其中, 属于 TEE 的内存为大于基地址 base3 的地址空间, 属于 MON 的内存为 base1–base3 的地址空间. 其中, base1–base2 保存安全世界上下文, base2–base3 保存普通世界上下文. 普通世界的内存属于安全域 REE.

内存模型 $M_a$ 建模为状态迁移系统, 由状态集合、事件集合、迁移函数和初始状态组成, 其定义如下.

**定义 2(内存模型).**

$$M_a=(\Sigma,E_a,\varphi_a,\sigma_0),$$

其中, $\Sigma$是系统状态的集合, $E_a$ 是事件的集合, $\varphi_a$: $E_a\rightarrow(\Sigma\times\Sigma)$是迁移函数, $\sigma_0\in\Sigma$是系统的初始状态.

(1) 内存模型中的系统状态记为$\sigma$, $\sigma\in\Sigma$, 包括寄存器以及内存, 即$\sigma=(R,M)$, 其中, $R$ 表示寄存器, $M$ 表示内存.

1) 寄存器 $R$=(GP,sysregs,PSTATE). 其中, GP 表示通用寄存器 X0-X30; sysregs 表示系统寄存器, 用于提供执行控制和系统配置; PSTATE 表示当前程序的状态信息;

2) 内存表示为附加地址位和物理地址到机器字的映射 *mem*: ($NS$×64 word)$\rightarrow W$, 其中, $NS$ 为附加的地址位, 指示该地址是安全的还是非安全的; 64 word 表示 64 位物理地址. 在抽象内存模型中, 虚拟地址到物理地址是一一映射的, 定义为 $map(\sigma,VA)=PA$, 其中, $VA=(NS,VA_w)$, $NS$ 指示 64 位虚拟地址 $VA_w$ 是安全的还是非安全的; $PA=(NS,PA_w)$, $NS$ 指示 64 位物理地址 $PA_w$ 是安全的还是非安全的.

(2) 内存模型中的事件 $E_a$ 包括 5 个事件, $E_a$={*world_switch*,*mem_read*,*mem_write*,*region_enable*,*region_disable*}.

1) *world_switch* 为世界切换事件. 抽象模型对基础模型[7]中的世界切换进行了改进, 基础模型中世界切换时没有保存和恢复上下文, 抽象模型增加了此功能. 本文根据 ATF 中世界切换的处理过程建模 *world_switch* 事件, 包含两步: 保存当前世界的处理器状态到内存中, 此过程建模为函数 *save_context*($\sigma$); 从内存中恢复要切换到的世界的处理器状态, 此过程建模为函数 *restore_context*($\sigma$). 如前文所述, 内存模型中设置固定的基地址作为保存两个世界上下文的内存起始位置. 两个世界切换时, 需要保存的寄存器包括 SCR 寄存器、程序状态寄存器(saved program status register, SPSR)和异常链接寄存器(exception link register, ELR), 在保存两个世界上下文的安全内存中, 保存寄存器 SCR、SPSR 和 ELR 的内存偏移量分别为 *CTX_SCR_EL*3、*CTX_SPSR_EL*3 和 *CTX_ELR_EL*3;

2) 其他事件的定义与文献[7]中的事件定义类似;

3) 事件的执行使用从一个状态到一个新状态的迁移来表示, 初始状态定义为$\sigma_0$, 从初始状态$\sigma_0$执行事件序列后获得的任何状态$\sigma$都是可达的, 记为 RE($\sigma$);

(3) 迁移函数$\varphi_a$:$E_a\rightarrow(\Sigma\times\Sigma)$表示一个系统状态通过执行事件集合 $E_a$ 中的事件迁移到另一个系统状态. 以

内存写事件 *mem_write*($\sigma$,*PA*,*val*)为例，其状态迁移过程为：系统状态$\sigma$执行内存写事件后得到系统状态$\sigma'$，$\sigma'$为将状态$\sigma$中物理地址值为 *PA* 处内存映射的内存值修改为 *val* 后得到的状态.

(4) 在初始状态$\sigma_0$中，寄存器 SCR，SPSR 和 ELR 的初始值均设置为 0x0000，通用寄存器的初始值也均设置为 0x0000; base2~base3 处内存设置为普通世界初始上下文，其他内存均初始化为 0x0000.

内存模型中基地址配置如下：保存上下文的内存基地址 base1, base2 和 base3 的值分别为 0x0000, 0x0200 和 0x0300. 保存寄存器值的内存偏移量 *CTX_SCR_EL*3, *CTX_SPSR_EL*3 和 *CTX_ELR_EL*3 的值分别为 0x0010, 0x0020 和 0x0030.

### 2.3 敌手模型

本节介绍敌手模型. 敌手在普通世界具有内核级权限，可以观察普通世界的寄存器和内存值，可以对普通世界的内存进行读写操作，但是不能观察安全世界的寄存器和内存值，也不可以对安全世界的内存进行任何的读写操作. 敌手模型是参数化的抽象模型，可以扩展敌手模型建模其他能力的敌手. 敌手模型 $A_a$ 的定义如下.

**定义 3(敌手模型).**

$$A_a=(O_a,E_a),$$

其中，$O_a$: $\Sigma\to\Sigma$表示敌手可以观察到的系统状态，即 $O_a(\sigma)=\sigma'$. 其中，$O_a(\sigma)$表示在状态$\sigma$中敌手观察到的系统状态，敌手观察到的系统状态包括普通世界的内存和寄存器的值. 事件 $E_a$={*adversary_read*, *adversary_write*}，其中，*adversary_read* 表示敌手对普通世界内存进行读操作，*adversary_write* 表示敌手对普通世界内存进行写操作.

本节以事件 *adversary_write* 为例，简要介绍敌手如何改变系统状态. 当处理器运行在普通世界时，敌手的能力有：观察普通世界中内存和寄存器的值、对普通世界的内存进行读写操作. 当 SCR 寄存器的 *NS* 位为 1 时，处理器运行在普通世界，事件 *adversary_write*($\sigma$,*PA*,*val*)可以将值 *val* 写入物理地址 $PA=(1,PA_w)$对应的内存里. 至此，敌手完成了对普通世界内存的写操作，篡改了 *PA* 处的内存. 事件 *adversary_read* 与之类似，故不再介绍.

### 2.4 正确性证明

模型正确性验证有两方面的作用，第 1 方面，可以检查 TrustZone 组件的正确性，比如通过验证虚拟地址映射到物理地址时不会溢出和非安全的虚拟地址不会映射到安全的物理地址上，可以保证内存隔离机制中执行地址映射的组件是正确的；通过验证安全内存不会被非安全的事务读取或修改，可以保证内存隔离机制中执行内存访问控制的组件是正确的. 第 2 方面，可以一定程度上保证模型与实际 TrustZone 内存隔离机制的一致性.

受篇幅限制，本文只给出地址映射正确性属性的两个定理.

**定理 1**. 物理地址值小于等于物理内存大小:

$$unit(addr\_val(map(\sigma,VA)))\leqslant memory\_size,$$

其中，函数 *unit* 将二进制地址值转换为十进制地址值，函数 *map* 将状态$\sigma$下的虚拟地址 *VA* 转换为物理地址，函数 *addr_val* 获取物理地址值, *memory_size* 表示物理地址空间的大小.

**定理 2**. 不存在非安全的虚拟地址映射到安全的物理地址:

$$\forall\sigma\ VA.RE(\sigma)\wedge((addr\_domain(VA)=1)\wedge(addr\_domain(map(\sigma,VA))=0))\to\text{False},$$

其中，函数 *addr_domain* 获取地址的附加位 *NS* 的值.

抽象模型的正确性在定理证明器 Isabelle/HOL 中进行了验证，验证结果表明，抽象模型满足正确性属性.

## 3 内存隔离机制的具体模型

本节对抽象模型进行精化建立具体模型，具体模型包括内存模型和敌手模型. 具体模型中的内存模型对

内存隔离机制的状态和组件进行了细粒度建模, 在抽象内存模型的基础上增加了 MMU 组件功能, 建模 MMU 的转换表和 TLB 机制. 下面首先对具体模型进行概述, 然后分别介绍状态精化、事件精化和具体模型的正确性证明.

### 3.1 具体模型概述

本节介绍对抽象模型进行精化后建立的具体模型, 该模型由内存模型、TrustZone 安全域、TrustZone 安全策略、等价关系和敌手模型组成. 具体模型 $TEE\_M_c$ 的定义如下.

**定义 4(具体模型).**

$$TEE\_M_c=(M_c,D,\rightsquigarrow,\sim,A_c).$$

具体模型 $TEE\_M_c$ 是一个五元组, 其中, $M_c$ 是内存模型, 内存模型描述与内存隔离相关的硬件组件和软件组件, 其中, 硬件组件包括 MMU 和 TZASC, 软件组件有 monitor, 相关组件的功能被建模为事件. 五元组中其他四元分别是在具体内存模型基础上定义的安全域集合 $D$、安全策略$\rightsquigarrow$、等价关系$\sim$和敌手模型 $A_c$.

具体模型中的内存模型 $M_c$ 将 MMU 组件的地址转换功能建模为事件 *mapping*, 包括对遍历转换表和 TLB 查找机制的建模. 具体模型中的内存模型 $M_c$ 定义内存、关键寄存器、转换表和 TLB 的状态, 并且对 MMU, TZASC 和 monitor 等组件的功能进行描述, 包括虚拟地址到物理地址的映射、内存划分、访问控制和世界切换等. 具体模型中的内存模型 $M_c$ 定义如下.

**定义 5(内存模型).**

$$M_c=(S,E_c,\varphi_c,s_0),$$

其中, $S$ 是系统状态的集合, $E_c$ 是事件的集合, $\varphi_c$: $(E_c\rightarrow S\times S)$是迁移函数, $s_0\in S$ 是系统的初始状态.

(1) 内存模型中的系统状态记为 $s$, $s\in S$. 系统状态 $s=(R,M,PG\_Table,TL\_Buffer)$, 其中, $R$ 表示寄存器, $M$ 表示内存, $PG\_Table$ 表示转换表, $TL\_Buffer$ 表示 TLB. $PG\_Table$ 和 $TL\_Buffer$ 是由于新增 MMU 组件而新引入的状态变量, 具体细节将在第 3.2 节详细介绍; 其他状态变量则与抽象内存模型 $M_a$ 的状态变量一致;

(2) $E_c$ 是事件的集合:

$$E_c=\{mapping,cworld\_switch,cmem\_read,cmem\_write,cregion\_enable,cregion\_disable\},$$

其中, 事件 *mapping* 是新增的事件, 其细节将在第 3.3 节介绍; 其他事件的定义与抽象内存模型 $M_a$ 中的事件定义类似.

具体内存模型 $M_c$ 精化了抽象内存模型 $M_a$ 的状态和事件, 下面分别介绍状态精化和事件精化.

### 3.2 状态精化

抽象模型中的内存模型 $M_a$ 的状态变量$\sigma$包括寄存器、内存, 通过用新的状态变量 $PG\_Table$、$TL\_Buffer$ 对$\sigma$进行扩展, 得到具体模型中的内存模型 $M_c$ 的状态变量 $s$:

$$s=\sigma+(PG\_Table,TL\_Buffer)=(R,M,PG\_Table,TL\_Buffer).$$

$PG\_Table$ 表示转换表, 是转换表条目的集合. TrustZone 中, 转换表分为安全世界的转换表和普通世界的转换表, 安全虚拟地址映射时遍历安全世界的转换表, 非安全虚拟地址映射时遍历普通世界的转换表. 转换表 $PG\_Table$ 的定义如下:

$$PG\_Table=(pb,PG\_index,T,PGF),$$

其中, $pb=(TTBR1,TTBR0)$表示转换表的基地址. 转换表基地址存储在转换表基址寄存器(translation table base registers, TTBRx)中, 其中, 内核空间的转换表基地址存储在 $TTBR1$ 中, 用户空间的转换表基地址存储在 $TTBR0$ 中. $PG\_index$ 表示虚拟地址索引值. $T\in\{4,16,64\}$表示转换表的转换粒度, 本文支持 4KB、16KB 和 64KB 这 3 种转换粒度, 不同的转换粒度有不同的地址索引值和地址偏移量 offset. $PGF$ 表示转换表描述符, 描述符由输出的物理地址 *base* 和内存属性信息标志位 *perms* 组成. 内存属性信息标志位 *perms* 由 3 个关键标志位组成, 分别为 *nG*、*AP* 和 *NS*, 其中, *nG* 表示是否全局有效; $AP=(rw,ro,no)$表示内存访问权限, 其中, *rw* 表示可读

可写，*ro* 表示只读，*no* 表示没有访问权限；*NS* 指示安全虚拟地址映射的物理地址属于安全内存还是非安全内存.

*TL_Buffer* 表示 TLB 条目的集合. *TL_Buffer*=(*asid*,*PG_index*,*base*,*I*)，其中，*asid* 表示地址空间 ID (address space ID，ASID)，将 ASID 与 TLB 条目关联，可以支持不同进程的地址转换缓存，避免进程切换造成的 TLB 频繁刷新；*PG_index* 表示虚拟地址索引值；*base* 表示输出的物理地址；*I* 表示属性信息，属性信息包括访问权限、安全状态等.

### 3.3 事件精化

具体模型中的内存模型增加了对 MMU 功能的描述，因而引入新事件 *mapping*，该事件是对抽象模型中地址映射过程的精化. 下面首先介绍 MMU 功能的建模方法[21]，然后分别介绍 TLB 和转换表功能的建模方法.

MMU 的地址转换功能由事件 *mapping* 描述，该事件以状态 *s*、转换粒度 *T* 和虚拟地址 *VA* 为参数进行地址转换，实现如下:

***mapping***(*s*,*T*,*VA*): =***case lookup***(*t*,*asid*,*VA*) of

*Miss*⇒(*Some*(*fst*(***translation***(*s*,*VA*,*T*))),*s*(|*TL_Buffer*:=(*t*∪*snd*(***translations***(*s*,*VA*,*T*)))|))

|*Hit x*⇒(***va_to_pa***(*VA*,*x*),*s*) (*函数 *va_to_pa* 使用命中的 TLB 条目 *x*，将虚拟地址 *VA* 转化成物理地址 *PA**)

|*Conflict*⇒(*None*,*s*)

*mapping* 根据查找 TLB 的结果做处理，如果未命中(*Miss*)，则遍历转换表；如果命中(*Hit*)，则将虚拟地址转换为物理地址；如果多条命中(*Conflict*)，则返回 *None*.

TLB 查找功能由函数 *lookup* 描述，该函数以 TLB 的条目集合 *t*、ASID 值 *asid* 和虚拟地址 *VA* 作为参数，返回查找 TLB 的结果，其实现如下:

***Lookup***(*t*,*asid*,*VA*): =**if** ***entry_set***(*t*,*asid*,*VA*)={·} **then** *Miss* (*函数 *entry_set* 用于遍历 TLB 条目集合*)

**else if** ∃*x*.***entry_set***(*t*,*asid*,*VA*)={*x*} **then** *Hit*

**else** *Conflict*

*lookup* 根据 *entry_set*(*t*,*asid*,*VA*)遍历 TLB 条目集合的结果做处理：如果结果为空，则返回 *Miss*；如果结果为某一 TLB 条目 *x*，则返回 *Hit*；否则返回 *Conflict*.

TLB 查找过程如图 2 所示.

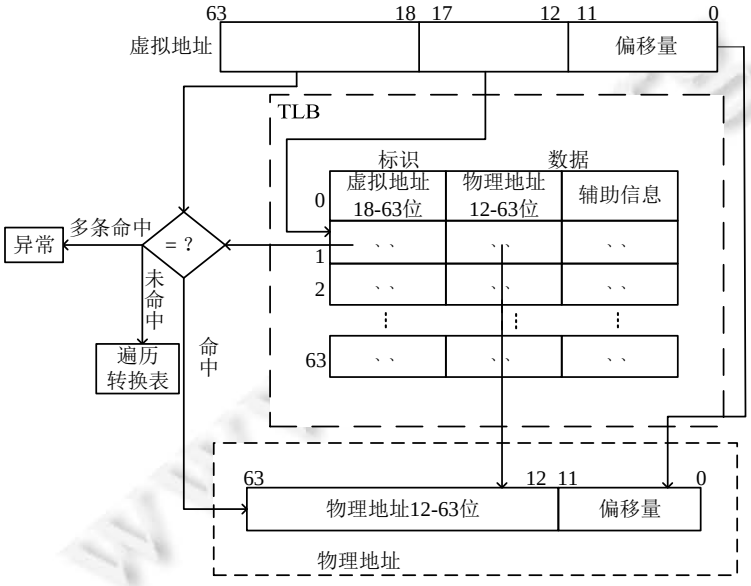


图 2 TLB 查找过程图

遍历转换表的功能由函数 *translation* 描述, 该函数以状态 *s*、虚拟地址 *VA* 和转换粒度 *T* 为参数, 返回完整的物理地址和新的 TLB 条目, 其具体实现如下:

***translation***(*s*,*VA*,*T*): =

**case** ***get_translation_table***(*s*,*pb*,*index*,*T*) of

(*函数 *get_translation_table* 用于遍历转换表, *index* 为虚拟地址的索引值*)

*fourKB base perms*⇒

((***get_paddr_four***(*base*,*offset*,*NS*,*perms*)),(***get_entry_four***(*asid*,*PG_index*,*offset*,*base*,*perms*)))

|*sixteenKB base perms*⇒

((***get_paddr_sixteen***(*base*,*offset*,*NS*,*perms*)),(***get_entry_sixteen***(*asid*,*PG_index*,*offset*,*base*,*perms*)))

|*sixtyfourKB base perms*⇒

((***get_paddr_sixtyfour***(*base*,*offset*,*NS*,*perms*)),(***get_entry_sixtyfour***(*asid*,*PG_index*,*offset*,*base*,*perms*))

*translation* 根据不同的粒度 *T* 计算出虚拟地址的索引值和偏移量, 通过 *TTBR*1 寄存器中的转换表基地址 *pb* 和虚拟地址中的索引值对转换表进行查找, 得到转换表描述符, 从中获取输出的物理地址 *base* 和内存属性信息标志位 *perms*. 使用函数 *get_paddr_*将 *base* 和地址偏移量 *offset* 组合得到 64 位物理地址, 并通过对虚拟地址的附加位 *NS* 和内存属性信息标志位 *perms* 进行比较, 得到物理地址附加位 *NS*, 从而得到完整的物理地址.函数 *get_entry_*以 ASID 的值 *asid*、虚拟地址索引 *PG_index*、地址偏移量 *offset*、输出的物理地址 *base* 和内存属性标志位 *perms* 为输入, 得到一个新的 TLB 条目. 遍历转换表的过程如图 3 所示.

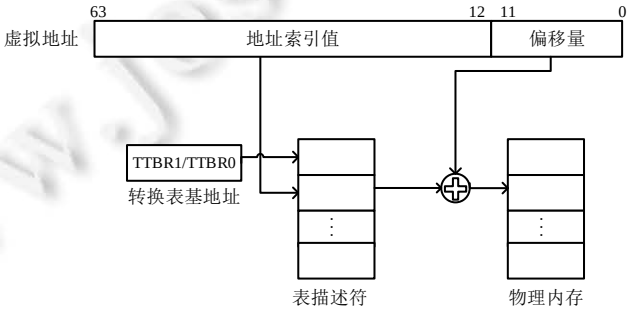


图 3 遍历转换表过程图

### 3.4 正确性证明

具体模型的正确性主要包括 MMU 功能的正确性、内存操作的正确性、TZASC 功能的正确性和世界切换的正确性. MMU 功能的正确性包括遍历转换表和 TLB 查找的正确性; 内存操作的正确性确保安全内存区域不被非法读取或修改; TZASC 功能的正确性保证安全内存区域不被非安全事务访问; 世界切换的正确性确保在世界切换过程中, 保存的上下文没有被篡改. 本文对以上正确性属性均进行了验证, 下面以MMU功能的正确性定义为例进行介绍.

MMU 功能的正确性定义包括遍历转换表和 TLB 查找两部分, 两者功能正确性的定义方法类似, 本节以 TLB 查找的正确性定义为例进行介绍. TLB 查找过程需要考虑查找结果是否正确, 以及删除 TLB 条目集合中的非待查找 ASID 对应的条目后查找结果是否发生改变. 对于 TLB 查找的 3 种不同结果, 其正确性定义方法相似, 本文以 TLB 命中时的正确性定义为例进行介绍, 如定理 3 和定理 4 所示.

**定理 3**. TLB 查找时, 若没有多条命中且没有未命中, 则 TLB 命中:

$\forall t\ asid\ VA.lookup(t,asid,VA)\neq Conflict\wedge lookup(t,asid,VA)\neq Miss\rightarrow\exists x\in t.lookup(t,asid,VA)=Hit\ x.$

定理3的含义为: 当查找TLB时不发生多条命中和未命中的情况时, 那么一定存在TLB条目 *x*, 使得TLB命中. 该定理表示 TLB 命中正确实现.

**定理 4**. 删除 TLB 中的非待查找 ASID 对应的条目后, 不改变 TLB 的命中情况:

$\forall t\ VA.asid_1 \neq asid_2 \wedge lookup(t,asid_2,VA)=Hit\ x \rightarrow lookup((asid\_invalidation(t,asid_1)),asid_2,VA)=Hit\ x$,

其中, 函数 $asid_invalidation(t,asid_1)$返回在 TLB 条目集合 $t$ 中删除所有 ASID 为 $asid_1$ 的条目后得到的新 TLB 条目集合. 定理 4 的含义为: $asid_1$ 和 $asid_2$ 为两个不同的 ASID, 并且以 $asid_2$ 和虚拟地址 $VA$ 为索引命中 TLB 条目 $x$, 当删除 TLB 中所有 ASID 为 $asid_1$ 的条目后, 以 $asid_2$ 和虚拟地址 $VA$ 为索引依然会命中 TLB 条目 $x$. 该定理表示删除 TLB 条目集合中的非待查找 ASID 对应的条目后, TLB 查找结果不发生改变.

在定理证明器 Isabelle/HOL 中, 对具体模型的验证结果表明, 具体模型满足上述正确性属性.

## 4 信息流安全性验证

本节首先提出内存隔离机制的两种安全策略, 定义信息流安全属性, 并证明抽象模型在两种安全策略下均满足信息流安全性; 然后证明抽象模型 $TEE\_M_a$ 和具体模型 $TEE\_M_c$ 之间的精化关系; 之后证明具体模型的信息流安全性; 最后给出验证结论并进行分析. 下面详细介绍抽象模型信息流安全性的证明、精化关系的证明、具体模型满足信息流安全性的精化证明以及验证结论.

### 4.1 安全策略及抽象模型的信息流安全性验证

本节介绍安全策略定义以及抽象模型中信息流安全属性的定义与证明.

#### 4.1.1 安全策略

TrustZone 访问控制模型允许 TEE 和 REE 间的双向信息流: 安全世界可以读写普通世界内存. 安全世界向普通世界内存进行写操作时, 可能导致 TEE 中敏感数据被有意或无意泄露到 REE 中, 破坏了 TEE 的机密性. 因此, 在安全性要求高的应用场景中, 如果要保证 TEE 中数据的机密性, 应禁止 TEE 到 REE 的直接信息流. 基于上述分析, 本文提出两种安全策略, 分别对应 TrustZone 的两种不同应用场景: 高安全要求应用场景、普通应用场景.

第 1 种安全策略 P1 不允许 TEE 向 REE 有直接信息流, 对应安全性要求高的应用场景, 要求保证 TEE 的机密性, 其定义如下.

P1: 1) $\forall d \in D.d \rightsquigarrow d$;

2) $\forall d \in D.d \rightsquigarrow MON$;

3) $\forall d \in D.MON \rightsquigarrow d$;

4) $TEE \not\rightsquigarrow REE$;

5) $REE \rightsquigarrow TEE$.

安全策略 P1 表示: (1) 允许安全域有流向自身的信息流; (2) 允许安全域到 MON 有信息流; (3) 允许安全域 MON 到任意安全域有信息流; (4) 不允许安全域 TEE 到安全域 REE 有信息流; (5) 允许安全域 REE 到安全域 TEE 有信息流.

第 2 种安全策略 P2 允许 TEE 向 REE 有直接信息流, 对应普通安全性要求的应用场景, 这是 TrustZone 访问控制模型的基础安全策略, 未在其上增加额外限制, 其定义如下.

P2: $\forall d_1, d_2 \in D.d_1 \rightsquigarrow d_2$.

上述定义表示: 3 个安全域既允许有流向自身的信息流, 也允许每个域到其他两个域有信息流.

#### 4.1.2 信息流安全属性定义

信息流安全属性是保证安全内存与非安全内存隔离的基础, 可以保证系统执行过程中不存在隐蔽的信息流通道. 信息流安全属性在等价关系的基础上定义, 本节首先介绍内存隔离机制模型中等价关系的定义, 然后介绍本文定义的信息流安全属性.

(1) 等价关系

等价关系包括状态等价和观察等价. 下面分别介绍状态等价和观察等价的定义.

**定义 6**(状态等价).

$s \sim d \sim t \equiv$ **if** $d=TEE$ **then** $(\forall v \geqslant base3.(M\ s)(0,v)=(M\ t)(0,v) \wedge$

(*current_world*(*s*)=*current_world*(*t*))∧(*current_world*(*s*)=*TEE*→(*R s*)=(*R t*)))

**else if** *d*=*REE* **then** (∀*v*.(*M s*)(1,*v*)=(*M t*)(1,*v*)∧

(*current_world*(*s*)=*current_world*(*t*))∧(*current_world*(*s*)=*REE*→(*R s*)=(*R t*)))

**else** (∀*v*∈(*base*1,*base*3).(*M s*)(0,*v*)=(*M t*)(0, *v*)∧(*R s*)=(*R t*))

上述定义的含义为: 状态 *s* 和 *t* 关于域 *d* 等价(*s*~*d*~*t*)的条件是:

- 若 *d* 为 *TEE*, 则 *s* 和 *t* 两个状态下安全世界的普通安全内存相同、当前处理器执行状态相同; 且若处理器运行在安全世界, 则两个状态下的寄存器也相同;
- 若 *d* 为 *REE*, 则 *s* 和 *t* 两个状态下非安全内存相同、当前处理器执行状态相同; 且若处理器运行在普通世界, 则两个状态下的寄存器也相同;
- 若 *d* 为 *MON*, 则两个状态下安全世界的安全内存相同且寄存器相同.

**定义 7**(观察等价).

$$s\triangleleft as\doteq d\doteq t\triangleleft bs\equiv\forall s\ t\ d\ as\ bs,\ (execute(s,as))\sim d\sim(execute(t,bs)).$$

上述定义的含义为: 两个状态 *s* 和 *t*, 如果 *s* 执行事件序列 *as* 后得到的新状态和 *t* 执行事件序列 *bs* 后得到的新状态对于域 *d* 等价, 则称该状态等价关系为观察等价, 记为 $s\triangleleft as\doteq d\doteq t\triangleleft bs$. 其中, *execute*(*s*,*as*)表示在状态 *s* 下执行事件序列 *as* 得到的新状态.

(2) 信息流安全属性

信息流安全属性包括无干扰属性、无泄露属性和无影响属性, 下面分别介绍这 3 个属性的形式化定义[23].

**定义 8**(无干扰属性).

$$noninterference\equiv\forall d\ as,\ (s_0\triangleleft as\doteq d\doteq s_0\triangleleft(ipurge(as,s_0,d))).$$

上述定义的含义为: 初始状态 $s_0$ 执行事件序列 *as* 后得到的状态和 $s_0$ 执行 *as* 经过清除操作后的事件序列得到的状态对于观察域 *d* 是观察等价的. 无干扰属性描述: 对于观察者来说, 不能分辨出执行和不执行被清除事件的区别, 即清除安全策略中不允许向观察域有信息流的事件不会影响系统执行过程中观察域观察到的状态.

**定义 9**(无泄露属性).

$$nonleakage\equiv\forall s\ t\ d\ as,\ RE(s)\wedge RE(t)\wedge(s\sim d\sim t)\wedge(s\sim(sources(s,as,d))\sim t)\rightarrow(s\triangleleft as\doteq d\doteq t\triangleleft as).$$

上述定义的含义为: 可达状态 *s* 和 *t* 关于观察域 *d* 等价, 并且关于安全策略允许对观察域 *d* 有信息流的域也等价, 则 *s* 和 *t* 执行事件序列 *as* 后得到的状态关于观察域 *d* 仍等价. 无泄露属性将执行相同事件序列的两个状态关联起来, 表示如果初始状态时没有信息泄露, 后续事件序列的执行也不会导致信息泄露.

**定义 10**(无影响属性).

$$noninfluence\equiv\forall s\ t\ d\ as,\ RE(s)\wedge RE(t)\wedge(s\sim d\sim t)\wedge(s\sim(sources(s,as,d))\sim t)\rightarrow(s\triangleleft as\doteq d\doteq t\triangleleft(ipurge(as,t,d))).$$

上述定义的含义为: 可达状态 *s* 和 *t* 关于观察域 *d* 是等价的, 并且关于安全策略允许对观察域 *d* 有信息流的域也是等价的, 则 *s* 执行事件序列 *as* 后得到的状态和 *t* 执行 *as* 经过清除操作后的事件序列得到的状态关于观察域 *d* 仍是等价的. 无影响属性是无干扰属性和无泄露属性的结合, 确保不允许向观察域有信息流动的事件不会对观察域的观察造成影响且观察域的机密数据不会泄露.

4.1.3 信息流安全性验证

本文前序工作[7]验证了基础模型在第 2 种安全策略下的信息流安全性, 因此本节仅简要介绍信息流安全属性的验证方法, 重点介绍在第 1 种安全策略下证明信息流安全属性时对抽象模型中的事件施加的限制.

本文通过验证系统各个执行步骤均满足展开定理[24]来证明抽象模型在两种安全策略下的信息流安全性. 首先, 通过验证抽象模型中所有事件均满足展开条件 *local_respect* 和 *weak_step_consistent*, 证明在抽象模型中展开条件成立; 然后, 根据展开定理(若两个展开条件成立, 则无影响属性成立)证明无影响属性; 之后, 根据无影响属性与无干扰和无泄露两个属性的蕴含关系证明这两个属性. 展开条件的形式化定义如下.

**定义 11**(***local_respect*** 条件).

$$local\_respect \equiv \forall s\ d\ e,\ RE(s) \wedge (domain\_of\_event(s,e) \not\rightsquigarrow d \rightarrow s \sim d \sim (exec\_event(s,e)),$$

其中，函数 *domain_of_event*(*s*,*e*)表示事件 *e* 的执行域，函数 *exec_event*(*s*,*e*)表示状态 *s* 执行事件 *e* 后得到的新状态. 上述定义的含义是：若状态 *s* 可达且若事件 *e* 的执行域不允许有流向观察域 *d* 的信息流，则状态 *s* 和执行事件 *e* 后得到的新状态关于观察域 *d* 是等价的.

**定义 12**(***weak_step_consistent* 条件**).

$$weak\_step\_consistent \equiv \forall s\ t\ d\ e,$$

$$RE(s) \wedge RE(t) \wedge (s \sim d \sim t) \wedge s \sim (domain\_of\_event(s,e)) \sim t \rightarrow (exec\_event(s,e) \sim d \sim (exec\_event(t,e)).$$

上述定义的含义是：若可达状态 *s* 和 *t* 关于观察域 *d* 以及事件 *e* 的执行域等价，则状态 *s* 和 *t* 执行事件 *e* 得到的新状态关于观察域 *d* 是等价的.

本文在有敌手模型的情况下验证了抽象模型在两种安全策略下满足信息流安全属性. 在安全策略 P1 下验证信息流安全性时，最初本文通过分析认为只有 TEE 写 REE 内存会引起 TEE 向 REE 的信息流，因此为满足安全策略 P1，在抽象模型的事件中限制 TEE 对 REE 的写操作，在此基础上验证信息流安全性. 然而在证明过程中发现，抽象模型不满足信息流安全性. 本文通过进一步分析发现，普通世界的内存区域使能 *region_enable* 和内存区域禁用 *region_disable* 两个操作也会引起 TEE 向 REE 的信息流. 之后通过在事件中对这两个操作也进行限制，证明了抽象模型的信息流安全性. 下面以内存区域使能 *region_enable* 为例说明对此类事件施加的限制，其定义如下：

***region_enable***(*s*,*PA*,*val*): =**if** (***current_world***(s)=*TEE*∧***addr_domain***(*PA*)=1) **then** (True,*s*)

**else *enableregion***(*s*,*PA*,*val*)

上述定义表示：若当前世界为 TEE 且物理地址 *PA* 的附加位 *NS* 的值等于 1 时，内存区域使能操作被禁止，系统状态 *s* 不发生改变；否则，调用 *enableregion* 函数完成内存区域使能操作.

### 4.2 精化关系定义与证明

本节参考文献[25]中的方法证明两个模型之间的精化关系. 后文中使用下标 *a* 和 *c* 分别表示抽象模型层次和具体模型层次. 在精化过程中，通过引入新的状态变量 $s_\Delta$，将抽象模型状态 $\sigma$ 转换为具体模型状态 *s*，即 $s=\sigma+s_\Delta$；通过引入新的事件 $\tau$，将抽象模型事件集合 $E_a$ 扩展为具体模型事件集合 $E_c$，即 $E_c=E_a\cup\{\tau\}$. 抽象模型和具体模型之间的精化关系 *R* 定义如下.

**定义 13(精化关系)**. 给定两个模型 $TEE\_M_a$ 和 $TEE\_M_c$，如果满足以下条件，则称具体模型 $TEE\_M_c$ 使用状态模拟关系 $\Phi: S\rightarrow\Sigma_a$ 和事件关系 $\Xi: E_c\rightarrow(E_a\cup\{\tau\})$ 精化了抽象模型 $TEE\_M_a$，记为 $TEE\_M_c \sqsubseteq_{\Phi\Xi} TEE\_M_a$.

(1) $TEE\_M_c$ 和 $TEE\_M_a$ 的初始状态满足精化关系，即 $\Phi(s_0)=\sigma_0$;

(2) $E_a$ 中每个事件都由 $E_c$ 中的一组事件精化并且 $\forall e\ s\ t,\ \Xi(e)\neq\tau\wedge(s,t)\in\varphi_c(e)\rightarrow(\Phi(s),\Phi(t))\in\varphi_a(\Xi(e))$;

(3) 新事件只改变精化中引入的新状态变量并且不影响 $TEE\_M_a$ 中的状态变量，即：

$$\forall e\ s\ t,\ \Xi(e)=\tau\wedge(s,t)\in\varphi_c(e)\rightarrow\Phi(s)=\Phi(t);$$

(4) 安全域集合不变并且精化得到的事件具有与抽象模型事件相同的执行域，即 $D_c=D_a$ 并且

$$\forall e\ s,\ \Xi(e)\neq\tau\rightarrow domain\_of\_event_c(s,e)=domain\_of\_event_a(\Phi(s),\Xi(e));$$

(5) 精化不改变安全策略，即 $\rightsquigarrow_c=\rightsquigarrow_a$;

(6) 具体模型的状态等价：$s\sim_c d\sim_c t=\Phi(s)\sim_a d\sim_a\Phi(t)\wedge s\sim_\Delta d\sim_\Delta t$，其中，$s\sim_\Delta d\sim_\Delta t$ 只考虑精化中引入的新状态变量 $S_\Delta$.

具体模型中，状态 *s* 执行事件 *e* 得到的结果状态的抽象是抽象模型中状态 $\sigma$ ($\sigma=\Phi(s)$)执行被精化的事件 $\Xi(e)$ 得到的结果状态的子集. 如果具体模型中的事件是新引入的事件，则其执行不影响抽象状态. 抽象模型的安全配置(安全域集合 *D*、安全策略 $\rightsquigarrow$、事件发起域、状态等价关系)在精化过程中保持不变，但是具体模型的状态等价关系还要求两个状态对新的状态变量等价，即 $\rightsquigarrow_\Delta$. 信息流安全性涉及状态和事件，精化关系表明，具体模型中状态-事件迹集合是抽象模型中状态-事件迹集合的子集，因此精化过程保持信息流安全性.

本文通过证明抽象模型和具体模型满足精化关系定义中的各项条件，证明了两者间的精化关系.

### 4.3 具体模型的信息流安全性验证

具体模型的信息流安全性仍通过展开定理证明, 由于引入新的状态变量和事件, 需要扩展抽象模型的展开条件. 然而, 因为具体模型和抽象模型间满足精化关系且抽象模型满足展开条件, 所以只需要在新的状态变量上证明展开条件, 就可以证明具体模型的信息流安全性.

具体模型中的事件在新状态变量上的 *local_respect* 条件和 *weak_step_consistent* 条件的定义方法类似, 下面以事件在新状态变量上的 *local_respect* 条件为例说明其定义方法.

**定义 14(事件在新状态变量上的 *local_respect* 条件)**.

$$local\_respect_{c\Delta}(e)\equiv\forall s\ d,\ RE(s)\wedge(domain\_of\_event_c(s,e))\not\sim_c d\rightarrow s\sim_c d\sim_c(exec\_event_c(s,e)).$$

具体模型信息流安全性的验证过程分为 4 步.

1) 证明新引入的事件在新状态变量上满足 *local_respect*;
2) 证明新引入的事件在新状态变量上满足 *weak_step_consistent*;
3) 证明非新引入的事件在新状态变量上满足 *local_respect*;
4) 证明非新引入的事件在新状态变量上满足 *weak_step_consistent*.

下面分别以新引入的事件 *mapping* 和非新引入的事件 *cmem_write* 为例, 介绍在具体模型中证明其满足展开条件的方法.

以新引入的事件 *mapping* 为例, 证明其在具体模型中满足展开条件, 只需证明该事件在新状态变量上满足 *local_respect* 条件和 *weak_step_consistent* 条件, 即:

**定理 5**. 事件 *mapping* 的 *local_respect* 条件:

$$\forall s\ d\ s'\ va\ T,\ RE(s)\wedge(domain\_of\_event_c(s,mapping(s,T,VA))\not\sim_c d)\wedge s'=snd(mapping(s,T,VA))\rightarrow s\sim_\Delta d\sim_\Delta s'.$$

**定理 6**. 事件 *mapping* 的 *weak_step_consistent* 条件:

$$\forall s\ t\ d\ s'\ t'\ VA\ T,\ RE(s)\wedge RE(t)\wedge s\sim_c d\sim_c t\wedge s\sim_c(domain\_of\_event_c(s,mapping(s,T,VA)))\sim_c t\wedge$$
$$s'=snd(mapping(s,T,VA))\wedge t'=snd(mapping(s,T,VA))\rightarrow s'\sim_\Delta d\sim_\Delta t'.$$

定理 6 包括 6 个前提: 状态 *s* 和 *t* 是可达的、*s* 和 *t* 关于域 *d* 等价、*s* 和 *t* 关于 *mapping* 的执行域等价、状态 *s′*是状态 *s* 执行 *mapping* 操作的后继状态、状态 *t′*是状态 *t* 执行 *mapping* 操作的后继状态. 其结论为: 状态 *s′*和 *t′*对于域 *d* 等价. 该定理的证明思路为: 首先, 根据观察域 *d* 的取值(*TEE*、*REE* 或 MON)分 3 种情况进行讨论; 然后, 在每种情况中, 根据 *mapping* 的执行域为 *TEE* 或 *REE* 分两种情况讨论, 通过证明在每种情况中 $s'\sim_\Delta d\sim_\Delta t'$都成立, 可证明该定理.

以非新引入的内存写事件 *cmem_write* 为例, 要证明其在具体模型中满足展开条件, 只需证明该事件在新状态变量上满足 *local_respect* 条件和 *weak_step_consistent* 条件, 即:

**定理 7**. 事件 *cmem_write* 的 *local_respect* 条件:

$$\forall s\ d\ s'\ PA\ val,\ RE(s)\wedge(domain\_of\_event_c(s,cmem\_write(s,PA,val))\not\sim_c d)\wedge$$
$$s'=fst(cmem\_write(s,PA,val))\rightarrow s\sim_\Delta d\sim_\Delta s'.$$

**定理 8**. 事件 *cmem_write* 的 *weak_step_consistent* 条件:

$$\forall s\ t\ d\ s'\ t'\ PA\ val,\ RE(s)\wedge RE(t)\wedge s\sim_c d\sim_c t\wedge s\sim_c(domain\_of\_event_c(s,cmem\_write(s,PA,val)))\sim_c t\wedge$$
$$s'=fst(cmem\_write(s,PA,val))\wedge t'=fst(cmem\_write(s,PA,val))\rightarrow s'\sim_\Delta d\sim_\Delta t'.$$

根据以上定理可得: 如果具体模型 $TEE\_M_c$ 是抽象模型 $TEE\_M_a$ 的精化, $TEE\_M_a$ 满足展开条件, 且具体模型中新引入的状态和事件也满足展开条件, 则具体模型满足信息流安全性. 以此为基础, 本文通过证明具体模型 $TEE\_M_c$ 在新状态变量上的展开条件, 证明了 $TEE\_M_c$ 的信息流安全性.

### 4.4 验证结论及分析

本文在定理证明器 Isabelle/HOL 中完成了 TEE 内存隔离机制的建模和验证. 整体验证工作的工作量为 10 人月. 建模部分工作量为 4 人月; 验证部分工作量为 6 人月, 其中, 事件内存写、内存区域使能和内存区域禁

用较为复杂, 其展开条件证明花费 2 人月; 其余事件的展开条件证明花费 4 人月. 在本文的验证工作中, 内存隔离机制的抽象模型和具体模型约 2 500 行代码; 证明部分使用结构化语言 Isar, 该部分代码约 5 800 行, 包含 350 条定理和引理, 这些定理和引理都得到了证明, 验证了内存隔离机制在两种安全策略下的信息流安全性.

如第 4.1.3 节所述, 本文在验证第 1 种安全策略下内存隔离机制的信息流安全性时发现, 除安全世界对普通世界的内存写操作会引起 TEE 向 REE 的信息流以外, 普通世界的内存区域使能和内存区域禁用也会引起 TEE 向 REE 的信息流: 当 TEE OS 执行普通世界的内存区域使能和禁用操作时, REE 可以通过检查其内存区域的使能情况获得 TEE 中信息. 因此, 在高安全要求的应用场景中, 若要保证 TEE 的机密性, 应在系统执行过程中禁止对普通世界内存区域的使能和禁用操作, 同时禁止安全世界对普通世界内存的写操作. 然而, 完全禁止该写操作可能会限制 TrustZone 在某些场景中的使用, 在这种场景中, TEE 用户可以增加对安全世界向普通世界写入数据的敏感性检查, 防止敏感数据经该操作泄露到普通世界. 这种方式可以在一定程度上提高 TEE 中数据的安全性. 本文建议 ARM 规范中增加一条对高安全要求 TEE 用户的使用提示: 如果要求保证 TEE 的机密性, 应禁止上述 3 种操作.

## 5 相关工作

本节介绍本文的相关工作, 包括可信执行环境的形式化验证、内存隔离机制的形式化验证和使用精化方法的形式化验证.

- 可信执行环境的形式化验证. 2015 年, 加州大学伯克利分校建立了 SGX 的形式化模型, 并提出了基于自动定理证明和信息流分析验证运行在 SGX 上的飞地程序机密性的工具 Moat[26]. 2017 年, 加州大学伯克利分校提出了可信抽象平台(trusted abstract platform, TAP)[27]. TAP 是飞地平台规范的形式化描述, 该团队验证了 TAP 的安全远程执行属性, 并通过精化证明了 Intel SGX 和 MIT Sanctum 实现了安全飞地. 2020 年, 加州大学伯克利分校提出了符合 POSIX 标准的保护飞地完整性免受恶意操作系统攻击的文件系统接口 BesFS[28], 并在定理证明器 Coq 中验证了 BesFS 实现符合其 API 规范, 证明了 BesFS 的安全性(safety). 2017 年, 康奈尔大学提出了一种验证硬件架构安全性的方法[29]. 利用该方法, 使用安全类型硬件描述语言 SecVerilogBL 实现了 ARM TrustZone 架构的多核原型, 通过静态信息流分析验证了该原型的安全性. 现有的可信执行环境验证工作侧重于验证 Intel SGX 架构 TEE 的安全性[26−28], 而验证 ARM TrustZone 架构 TEE 的工作[29]仅考虑了其硬件实现. 本文则是对 TrustZone 架构内存隔离机制的硬件和软件进行了细粒度的建模和验证;
- 内存隔离机制的形式化验证. 2016 年, 瑞典皇家理工学院提出了使用直接分页的 ARMv7 处理器内存虚拟化平台的设计、实现和验证[5], 使用定理证明器 HOL4, 基于 ARMv7 MMU 模型精化证明了 hypervisor 的 MMU 完整性、内存隔离和信息流正确性. 2021 年, 哥伦比亚大学使用 Coq 精化验证了经改造的 Linux KVM hypervisor[6], 将 KVM 分解为一个小的对 KVM 其余部分实施访问控制的可验证核心 KCore 和一组不可信服务, 在验证 KCore 功能正确性的基础上, 证明整个 KVM 实现了 VM 的私有内存隔离. 2018 年, 法国里尔大学使用 Coq 验证了基于 MMU 的抽象内存管理模型[10], 将隔离所需的系统功能正确性属性描述为一致性属性, 通过证明该属性, 验证其抽象内存管理模型能够保证进程的物理内存隔离. 2020 年, 首都师范大学验证了 ARMv8 TrustZone 架构内存隔离机制的安全性[7], 建立了包含内存隔离主要硬件组件和世界切换功能的基础模型, 验证了 TrustZone 内存管理的正确性和内存隔离机制的信息流安全性. 大部分内存隔离机制验证工作[5−7,10]未完整建模内存隔离组件(如 TLB)且未考虑复杂信息流安全属性. 文献[7]是本文的前序工作, 其建立了 TrustZone 内存隔离机制的基础模型, 但是未对该机制涉及的组件进行完整的细粒度建模;
- 使用精化方法的形式化验证. 2013 年, 澳大利亚新南威尔士大学对 seL4 微内核进行了精化验证[8,30,31], 将 seL4 分成抽象层、实现层和 C 代码层这 3 个规范层次, 为每个规范层次建立形式化模型,

通过精化关系证明了抽象层建立的模型所满足的属性也适用于另外两层建立的模型[31], 最终证明了seL4的信息流安全性[30]. 2016年, 耶鲁大学提出了验证并发操作系统内核正确性的CertiKOS架构[9], 通过定义一系列逻辑抽象层和上下文精化关系来实施验证, 并利用该架构, 在Coq中通过约40层抽象证明了mC2内核的功能正确性. 该团队还验证了mCertiKOS内核的安全性[11], 通过在不同的抽象层使用不同的观察函数, 并使用模拟关系连接跨抽象层的安全证明, 验证了由C语言和汇编语言编写的mCertiKOS的无干扰属性. 2018年, 华盛顿大学提出了用于验证系统接口和实现的无干扰属性的框架Nickel[32], 引入了一种保持无干扰属性的精化形式, 调用Z3 SMT求解器, 通过检查展开和精化条件, 验证实现的无干扰属性, 并使用此框架验证了NiStar、NiKOS和ARINC 653标准. 2017年, 北京航空航天大学提出了符合ARINC 653标准的隔离内核安全性的精化验证方法[25], 使用逐步精化框架为隔离内核建立了两层功能规范, 在Isabelle/HOL中证明了规范的安全性, 且在验证和代码审查过程中发现了ARINC 653标准和实现中的6个可能导致信息泄露的安全漏洞. 大部分精化验证工作[8,9,11,25,30,31,32]都是对操作系统内核进行分层建模, 通过证明各层次模型间满足精化关系, 验证其正确性或安全性. 而本文采用与文献[25]类似的精化方法, 验证了涉及硬件和软件两方面的TEE内存隔离机制的安全性.

## 6 结束语

本文提出一种基于精化的TEE内存隔离机制安全性验证方法, 使用此方法对ARMv8 TrustZone架构内存隔离机制的安全性进行了验证. 本文建立了内存隔离机制的抽象模型和具体模型, 通过对TrustZone访问控制模型的分析, 制定了两种安全策略. 在证明针对第1种安全策略模型的安全性时发现: 仅对安全世界向普通世界的内存写操作施加限制并不能保证TEE的机密性, 普通世界内存区域的使能和禁用操作仍会引起TEE向REE的信息流. 在限制上述3种操作后, 证明了在该安全策略下模型满足信息流安全属性. 最终, 利用不同的限制条件, 本文证明了TEE内存隔离机制在两种安全策略下的信息流安全性. 通过分析验证结果, 本文建议ARM规范中增加对高安全要求用户的TrustZone使用提示: 若要保证TEE中敏感数据的机密性, 应禁止安全世界向普通世界的内存写操作, 以及普通世界内存区域的使能和禁用操作.

本文精化建立的TEE内存隔离机制具体模型中增加了MMU功能, 建模了MMU中的转换表和TLB, 但是未对cache进行建模; 本文建立了敌手模型, 但敌手模型中未考虑具有侧信道攻击能力的敌手. 未来的工作是在现有模型的基础上增加精化层次, 建模cache组件, 并在敌手模型中加入侧信道攻击能力, 以便更加完整地验证TrustZone内存隔离机制的安全性.

**References**:

[1] Sabt M, Achemlal M, Bouabdallah A. Trusted execution environment: What it is, and what it is not. In: Proc. of the 14th IEEE Int'l Conf. on Trust, Security and Privacy in Computing and Communications (TrustCom). IEEE, 2015. 57−64. [doi: 10.1109/Trustcom.2015.357]

[2] Pinto S, Santos N. Demystifying ARM TrustZone: A comprehensive survey. ACM Computing Surveys (CSUR), 2019, 51(6): 1−36. [doi: 10.1145/3291047]

[3] Costan V, Devadas S. Intel SGX explained. IACR Cryptology. ePrint Archive, 2016, 2016(86): 1−118.

[4] Göttel C, Pires R, Rocha I, Vaucher S, Felber P, Pasin M, Schiavoni V. Security, performance and energy trade-offs of hardware-assisted memory protection mechanisms. In: Proc. of the 37th IEEE Symp. on Reliable Distributed Systems (SRDS). IEEE, 2018. 133−142. [doi: 10.1109/SRDS.2018.00024]

[5] Guanciale R, Nemati H, Dam M, Baumann C. Provably secure memory isolation for Linux on ARM. Journal of Computer Security, 2016, 24(6): 793−837. [doi: 10.3233/JCS-160558]

[6] Li SW, Li X, Gu R, Nieh J, Hui JZ. A secure and formally verified Linux KVM hypervisor. In: Proc. of the 42th IEEE Symp. on Security and Privacy (S&P). IEEE, 2021. 1782−1799. [doi: 10.1109/SP40001.2021.00049]

[7] Ma YW, Zhang QY, Zhao SJ, Wang GH, Li XM, Shi ZP. Formal verification of memory isolation for the TrustZone-based TEE. In: Proc. of the 27th Asia-Pacific Software Engineering Conf. (APSEC). IEEE, 2020. 149−158. [doi: 10.1109/APSEC51365.2020.00023]

[8] Klein G, Andronick J, Elphinstone K, Murray T, Sewell T, Kolanski R, Heiser G. Comprehensive formal verification of an OS microkernel. ACM Trans. on Computer Systems (TOCS), 2014, 32(1): 1−70. [doi: 10.1145/2560537]

[9] Gu R, Shao Z, Chen H, Wu X, Kim J, Sjoberg V, David C. CertiKOS: An extensible architecture for building certified concurrent OS kernels. In: Proc. of the 12th USENIX Symp. on Operating Systems Design and Implementation (OSDI). Savannah: USENIX Association, 2016. 653−669.

[10] Jomaa N, Nowak D, Grimaud G, Hym S. Formal proof of dynamic memory isolation based on MMU. Science of Computer Programming, 2018, 162(34): 76−92.

[11] Costanzo D, Shao Z, Gu R. End-to-end verification of information-flow security for C and assembly programs. In: Proc. of the 37th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). Santa Barbara: ACM, 2016. 648−664. [doi: 10.1145/2980983.2908100]

[12] ARM Ltd. ARM architecture reference manual, ARMv8, for ARMv8—A architecture profile. 2019.

[13] Ning ZY, Zhang FW. Ninja: Towards transparent tracing and debugging on ARM. In: Proc. of the 26th USENIX Security Symp. (USENIX Security 2017). Vancouver: USENIX Association, 2017. 33−49.

[14] Ngabonziza B, Martin D, Bailey A, Cho H, Martin S. TrustZone explained: Architectural features and use cases. In: Proc. of the 2nd IEEE Int'l Conf. on Collaboration and Internet Computing (CIC). IEEE, 2016. 445−451. [doi: 10.1109/CIC.2016.065]

[15] Bhattacharjee A. Large-reach memory management unit caches. In: Proc. of the 46th Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO). New York: ACM, 2013. 383−394. [doi: 10.1145/2540708.2540741]

[16] Wirth N. Program development by stepwise refinement. Communications of the ACM, 1971, 14(4): 221−227. [doi: 10.1145/362575.362577]

[17] Back R. A calculus of refinements for program derivations. Acta Informatica, 1988, 25(6): 593−624. [doi: 10.1007/BF00291051]

[18] Lynch N, Vaandrager F. Forward and backward simulations. Information and Computation, 1995, 121(2): 214−233. [doi: 10.1006/inco.1995.1134]

[19] Zhao Y, Yang ZB, Ma DF. A survey on formal specification and verification of separation kernels. Frontiers of Computer Science, 2017, 11(4): 585−607. [doi: 10.1007/s11704-016-4226-2]

[20] Nelson L, Bornholt J, Krishnamurthy A, Torlak E, Wang X. Noninterference specifications for secure systems. ACM SIGOPS Operating Systems Review, 2020, 54(1): 31−39. [doi: 10.1145/3421473.3421478]

[21] Ma YW. Formal verification of memory isolation for the TrustZone-based TEE [MS. Thesis]. Beijing: Capital Normal University, 2021 (in Chinese with English abstract).

[22] Nipkow T. Programming and proving in Isabelle/HOL. Technical Report, University of Cambridge, 2013.

[23] Von Oheimb D. Information flow control revisited: Noninfluence=noninterference+nonleakage. In: Proc. of the 9th European Symp. on Research in Computer Security (ESORICS). Berlin: Springer, 2004. 225−243. [doi: 10.1007/978-3-540-30108-0_14]

[24] Mantel H. Unwinding possibilistic security properties. In: Proc. of the 6th European Symp. on Research in Computer Security. Berlin: Springer, 2000. 238−254. [doi: 10.1007/10722599_15]

[25] Zhao YW, Sanán D, Zhang FY, Liu Y. Refinement-based specification and security analysis of separation kernels. IEEE Trans. on Dependable and Secure Computing (TDSC), 2017, 16(1): 127−141. [doi: 10.1109/TDSC.2017.2672983]

[26] Sinha R, Rajamani S, Seshia S, Vaswani K. Moat: Verifying confidentiality of enclave programs. In: Proc. of the 22nd ACM SIGSAC Conf. on Computer and Communications Security (CCS). Denver: ACM, 2015. 1169−1184. [doi: 10.1145/2810103.2813608]

[27] Subramanyan P, Sinha R, Lebedev I, Devadas S, Seshia SA. A formal foundation for secure remote execution of enclaves. In: Proc. of the 24th ACM SIGSAC Conf. on Computer and Communications Security (CCS). Dallas: ACM, 2017. 2435−2450. [doi: 10.1145/3133956.3134098]

[28] Shinde S, Wang S, Yuan P. BesFS: A POSIX filesystem for enclaves with a mechanized safety proof. In: Proc. of the 29th USENIX Security Symp. (USENIX Security 2020). Savannah: USENIX Association, 2020. 523−540.

[29] Ferraiuolo A, Xu R, Zhang D, Myers AC, Suh GE. Verification of a practical hardware security architecture through static information flow analysis. In: Proc. of the 22nd Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). Xi'an: ACM, 2017. 555−568. [doi: 10.1145/3037697.3037739]

[30] Murray T, Matichuk D, Brassil M, Gammie P, Bourke T, Seefried S, Lewis C, Xin G, Klein G. seL4: From general purpose to a proof of information flow enforcement. In: Proc. of the 34th IEEE Symp. on Security and Privacy (S&P). IEEE, 2013. 415−429. [doi: 10.1109/SP.2013.35]

[31] Klein G, Norrish M, Sewell T, Tuch H, Winwood S, Elphinstone K, Heiser G, Andronick J, Cock D, Derrin P, Elkaduwe D, Engelhardt K, Kolanski R. seL4: Formal verification of an OS kernel. In: Proc. of the 22nd ACM SIGOPS Symp. on Operating Systems Principles (SOSP). Montana: ACM, 2009. 207−220. [doi: 10.1145/1629575.1629596]

[32] Sigurbjarnarson H, Nelson L, Castro-Karney B, Bornholt J, Torlak E, Wang X. Nickel: A framework for design and verification of information flow control systems. In: Proc. of the 13th USENIX Symp. on Operating Systems Design and Implementation (OSDI). Carlsbad: USENIX Association, 2018. 287−305.

**附中文参考文献:**

[21] 马雨薇. 可信执行环境内存隔离机制的形式化验证 [硕士学位论文]. 北京: 首都师范大学, 2021.

**靳翠珍**(1992－), 女, 硕士生, CCF 学生会员, 主要研究领域为形式化验证, 系统安全.

**王国辉**(1984－), 男, 博士, 高级实验师, CCF 专业会员, 主要研究领域为形式化验证, 高可靠嵌入式系统.

**张倩颖**(1986－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为形式化验证, 系统安全, 操作系统.

**施智平**(1974－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为形式化验证, 视觉信息处理.

**马雨薇**(1995－), 女, 硕士, 主要研究领域为形式化验证, 系统安全.

**关永**(1966－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为形式化验证, 高可靠嵌入式系统.

**李希萌**(1987－), 男, 博士, 讲师, CCF 专业会员, 主要研究领域为形式化验证, 区块链系统可靠性, 信息流安全.