华中科技大学

博士学位论文

基于虚拟机架构的可信计算环境构建机制研究

姓名：程戈

申请学位级别：博士

专业：计算机系统结构

指导教师：金海

2010-12-23

摘 要

通过软件和可信硬件平台相结合的方式构建可信计算环境，为解决计算机安全面临的挑战提供了新的途径。基于虚拟机架构构建可信计算环境，能为上层应用提供强有力的安全保证。

可信计算领域，国内外都处于技术超前于理论的状况，这使基于虚拟机架构的可信计算环境构建缺乏理论指导。现有的研究将虚拟机监控器作为信任链中的一环，这使现有的操作系统无法支持应用程序级的完整性度量和认证。同时，基于虚拟机架构的计算环境大多采用通用的虚拟机监控器，这使基于虚拟机架构构建可信计算环境具有较大虚拟化开销。结合可信计算在虚拟机系统中实施强制访问控制策略，可信虚拟域能够管理通过授权的公开通道传递的信息流，却不能够控制通过隐形通道传递的具有潜在风险的信息流。针对这些问题，需要对基于虚拟机架构的可信计算环境进行构建理论和构建机制研究。

信任链模型和可信虚拟域隔离模型是基于虚拟机架构可信计算环境构建的支持理论。基于实体间的依赖关系，提出一个具有普遍意义的信任链模型。该模型给出可信状态、信任根、信任度量的形式化定义，并对基于静态可信度量根和动态可信度量根的信任链进行统一建模。该模型为评估现有的可信计算环境提供理论依据，并为构建基于虚拟机架构的可信计算环境提供理论支撑。

针对可信虚拟域中的隐形流问题，提出一种在可信虚拟域间用于控制隐形流的优先中国墙模型。优先中国墙模型具有和传统中国墙模型不同的访问规则，该模型通过建立联盟关系实现对利益冲突关系的动态扩展，从而构建访问区域。该模型用于隐形流控制时，既拥有类似中国墙策略依据主体访问需要进行选择的灵活性，又拥有类似格策略能够将客体分成不同的访问集便于实施的优点。

基于上述模型，提出三种基于虚拟机架构的可信计算环境构建机制：基于虚拟机架构的透明信任链机制、基于轻量级虚拟化的动态可信执行环境构建机制和可信虚拟域间隐形流控制机制。基于虚拟机架构的透明信任链机制能够构建对操作系统

透明的应用级信任链，可以在用户期望的可信计算环境遭到破坏时保护其敏感数据，并使普通的商用平台具有类似 IBM 4758 安全协处理器平台的完整性和机密性特征。

基于轻量级虚拟化的动态可信执行环境构建机制利用动态可信度量根和硬件虚拟化技术在运行的操作系统下插入一个轻量级的虚拟机监控器，并使用该监控器为目标程序选择的内存页提供隔离的安全运行空间，从而实现细粒度的访问控制和灵活的内存保护。该机制是基于动态可信度量根技术的首个虚拟化实现，它拥有更小的可信基及启动和运行开销，并可以保护现有商用操作系统上很大范围的遗留程序。

可信虚拟域间隐形流控制机制在每个节点建立从信任根到虚拟机监控器的信任链，然后相互验证并扩展信任链于整个分布式系统，从而形成联合可信基。该机制在联合可信基上实施优先中国墙策略，依据用户的安全需求消除可信虚拟域间的隐形流。可信虚拟域间隐形流控制机制能满足虚拟机架构中企业的高安全需要，确保敏感的信息不会泄露给竞争对手。

关键词：虚拟机架构，信任链模型，动态可信度量根，可信执行环境，隐形流控制

Abstract

Trusted computing environment provides a new arena to address the challenges of computer security by combining software and trusted computing hardware. Virtualization based trusted computing environment further offers powerful security protection for upper applications.

However, the development of current trusted computing technology advances its theoretical study, which makes virtualization based trusted computing environment lack of theoretical foundation. Most existing research work consider virtual machine monitor (VMM) as part of trust chain. As a result, existing operating systems cannot support application-level integrity protection. Meanwhile, virtualization-based trusted computing environment generally uses a commodity VMM, which introduces significant performance overhead for building the trusted computing environment. By combining trusted computing and mandatory access control (MAC) policies in virtual machine systems, trusted virtual domain (TVD) can manage authorized overt information flow, however it cannot control the potential risks of covert channels. To solve these problems, this dissertation presents the research on virtualization based trusted computing environment from theoretical and enabling mechanism aspects.

This dissertation first defines the chain of trust model in trusted computing environment and isolation model in trusted virtual domains. The chain of trust model in this dissertation is universal, which offers a formal definition of trusted state, trusted root and trust measurement, and unified modeling for DRTM (dynamic root of trust for measurement) and SRTM (static root of trust for measurement), with the assumption that the authenticity of an entity can measure its behavior without any loss. This model also provides a theoretical basis for assessing the existing trusted computing environment, and offers theoretical support for the follow-up research on how to build a more reasonable virtualization based trusted computing environment. The isolation model in trusted virtual domains (Priority Chinese Wall -- PCW) prevents covert flow through careful resource management, and enables users to mitigate remaining covert channels through configuration optionswhile preserving the freedom of choice characteristic of the

traditional Chinese Wall policy. It also partitions VM labels into different ranges, as the Caernarvon model does.

By following these theoretical principles, this dissertation further presents three t mechanisms to build virtualization based trusted computing environment: transparent chain of trust infrastructure (TCT), lightweight virtualization based dynamic trusted execution environment--Cherub, and covert flows confinement (CFC) mechanism in trusted virtual domains (TVD).

TCT aims to extend TCG chain of trust to application layer in virtualization based environment, but maintain the transparency to operating system. It can protect user's sensitive data when their integrity of the environment is broken. TCT solves the problem that the application-level chain of trust is too long in traditional virtualization based environment. Furthermore, it enables commodity platforms the capability to protect the integrity and confidentiality of applications and data, which is similar to IBM 4758 coprocessor equipped platforms.

Cherub leverages the dynamic root of trust and hardware virtualization technology to insert a lightweight virtual machine monitor (LVMM) under a running operating system. Through the LVMM, Cherub is able to isolate a trusted execution environment for the memory pages selected by the target process to achieve fine-grained access control and flexible memory protection. Cherub is the first VMM that leverages late launch based dynamic root of trust, which has smaller scale of code and runtime overhead than legacy VMMs. It can protect a wide range of legacy programs in existing commercial operating systems.

Covert flow confined mechanism (CFC) establishes the chain of trust from the root of trust to VMM on each node, and then expands the trust chain to distributed enviornmentwith mutual verification. With this, a joint trust base can be achieved. CFC effectively confines covert flows in TVD by enforcing the PCW, which assures that critical information on such systems would not be leaked to the competitors and satisfies enterprise level security requirement.

Key words: Virtual Machine Architecture, Chain of Trust Model, Lightweight Virtual Machine Monitor, Trusted Execution Environment, Covert Flow Confinement

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到，本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　月　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保密□ ，在_____年解密后适用本授权书。

　　不保密□。

（请在以上方框内打“ √ ”）

学位论文作者签名：　　指导教师签名：

日期：　年　月　日　　日期：　年　月　日

1 绪论

本章首先介绍虚拟机架构下可信计算环境构建的研究背景，然后分析当前国内外研究的现状和不足，接下来给出本文的研究内容和主要贡献，最后介绍本文的组织结构。

1.1 研究背景

1.1.1 可信计算

随着信息技术的发展，现代社会越来越依赖于计算机系统。特别是近年来，在互联网技术的推动下，计算机越来越多地应用到社会政治、经济、教育和军事等领域中，使计算平台的安全性变得愈发的重要。然而自从计算机问世以来，计算机安全问题就一直伴随着计算机的发展而存在。近三十年来，其造成的损失也越来越严重。造成这种情况的一个重要原因是传统的安全防护方式和软件的固有缺陷不足以防御日益增多的计算机安全问题：第一，防火墙、入侵监测和病毒防范是构成传统信息安全系统的主要技术手段，这些技术手段是一种事后响应方式，即在攻击发生后或是进行中，通过对已发生过的滞后信息进行分析来判定是否存在攻击，从而进行相应的响应或是防护。面对当今日趋复杂和变化多端恶意攻击手段，这些事后相应的传统防护手段往往无力应对新的攻击方式；第二，现有平台架构是开放式的，计算机资源可任意被使用，尤其是执行代码可被任意修改。因此，在现有软件架构下，恶意程序很容易植入软件系统中。如果缺乏相关硬件的支持，仅仅依靠软件本身是不能完全检测出恶意代码，因为所有试图通过软件检测恶意代码的方法都无法证明检测软件自身是安全的。

要克服这些问题，可信计算提供一个新的思路: 从接入端增强系统的安全性，使系统每个接入端的计算平台都具有一定的物理保护，并在这样的平台上通过软硬件结合的方式构建可信的计算环境。可信计算环境可以确保其上进行的计算具有某些特性，例如使用可信计算环境保证其中运行程序和数据的完整性、真实性、机密性、

可控性等。利用可信计算环境提供的这些特性可以弥补仅依靠软件安全防护方式带来的不足，从而更好的解决计算机安全面临的问题和挑战。

可信计算为解决计算机的安全问题带来了新的希望，然而在中文计算机语境中，关于“可信”却有着不同的定义，在中文文献中使用最多的有以下几种：

可信计算组织（Trusted Computing Group）TCG[1]用实体行为的预期性来定义可信：如果一个实体的行为总是以预期的方式，朝着预期的目标，则该实体是可信的。

ISO/IEC 15408[2]标准定义可信为：参与计算的组件、操作或过程在任意的条件下是可预测的，并能够抵御病毒和物理干扰。

我国著名的信息安全专家屈延文教授[3]把“可信”解释为“可信性是考察行为预期性的满足，这种预期性满足是在多主体多行为范畴内，实现对行为性质、行为输入输出、行为过程、行为属性等方面符合必须遵守的要求、约定、规定、规则、法律的满足性认识与评价。”

其它的一些解释还有：当第二个实体按着第一个实体的期望行为时，第一个实体可假设第二个实体是可信的[4]；可信≈安全+可靠，可信计算系统是能够提供系统的可靠性、可用性、信息和行为安全性的计算机系统[5]等等。

从中文语义上看，这些定义是混乱的甚至是自相矛盾。例如：按照 TCG 的定义，一个恶意软件在期望的时间内做出了期望的危害行为，那么它是可信的，而依照 ISO/IEC 15408 标准定义却是不可信的。而实际上，上述的定义都是正确的，是对“信任”从不同的方面给出了定义。

信任(Trust)是一个很复杂的概念，而且不同领域的研究者对此也做出了不同的定义。这一概念已经长期存在于人类社会中，但是计算机科学领域对信任的研究才刚刚起步，大多借鉴了心理学、社会学和经济学等其他科学研究领域的成果，不同领域的研究者其侧重点是不相同的，有些着重信任的形式化描述，有些着重信任的特征研究，有些着重信任在实际系统中应用研究，这就造就了信任定义的多样化，甚至是相互冲突的。

实际上，如果追溯上述可信定义历史起源，可以看到这些定义对应着以下三种不同的研究背景：可信赖计算（Dependable Computing）、安全计算(Security Computing)

和信任计算(Trusted Computing)。可信赖计算源自于早期的容错计算：可信赖计算主要针对元器件、系统和网络，在设计、制造、运行和维修全过程中出现的各种非恶意故障，进行故障检测、故障诊断、故障避免、故障容许，使系统能够高可靠、高可用。安全计算主要针对系统和网络运行过程中的恶意攻击，同可信赖计算的不同在于它们针对的故障不同，安全计算主要针对的是人为的恶意攻击，而可信赖计算针对的是非恶意的攻击[6]。而信任计算源自于早期的安全硬件设计，基本思想就是在假定真实性可以用于度量信任并而不考虑度量中的损失，给出了一个信任在实体间传递的方法--在计算机系统中首先建立一个信任根，再建立一条信任链，一级测量认证一级，一级信任一级，把信任关系扩大到整个计算机系统，从而确保计算机系统的可信。

可信赖计算、安全计算和信任计算在研究内容上有一定的交叉，并且国内的不少文献都将 Dependable Computing 和 Trusted Computing 统一翻译成可信计算。本文认为从广义上讲可信计算内涵和外延将包括可信赖计算、安全计算和信任计算的研究内容。而本文的研究更加偏重于 TCG 给出的可信定义，如果没有特别指出，在本文中可信计算是指系统提供的计算服务能够满足需求者对计算的期望，并且系统具有能够提供证明其上计算具有真实性、完整性、机密性和可控性的能力。

同时本文认为可信计算的目的是通过软硬件相结合的方式使其上进行的计算具有某些特性，并利用这些特性来弥补仅依靠传统安全防护方式的不足，从而更好地解决计算机安全面临的问题和挑战。但可信计算不能等同于绝对安全，可信计算不能解决所有的安全问题，可信计算只是提供了一种加强系统安全的方式，能够结合传统的安全技术来增强系统的安全性。

1.1.2 虚拟化技术

虚拟化技术起源于 20 世纪 60 年代，IBM 将虚拟化技术应用于大型机，允许用户在一台主机上运行多个操作系统以充分利用昂贵的资源。随后，大型机上的虚拟化技术开始向小型机或 UNIX 服务器上移植，HP、SUN 也跟随 IBM 在自己的 RISC

服务器上提供了虚拟化技术，但由于真正使用大型机和小型机的用户还是少数，加上各家产品和技术之间并不兼容，虚拟化技术仍旧不太被公众所关注。

近年来，随着计算系统的资源规模不断扩展、处理能力快速增强、资源种类日益丰富、应用需求灵活多样。特别是随着 x86 处理器性能的提升和应用普及，以及多核技术的发展，使虚拟化技术成为商业和学术界关注的热点。现在，Intel 和 AMD 都在硬件层次上对虚拟化提供了支持，例如 Intel 的 VT 技术[7][8]和 AMD 的 AMD-V[9] 技术。x86 平台可以提供便宜的、高性能和高可靠的服务器，因此随着虚拟化技术在 x86 平台上的逐步实现，虚拟化技术开始向人们展示了其广阔的应用前景：虚拟化计算系统能够动态组织多种计算资源，隔离具体的硬件体系结构和软件系统之间的紧密依赖关系，实现透明化的可伸缩计算系统架构，从而灵活构建满足多种应用需求的计算环境，提高计算资源的使用效率，发挥计算资源的聚合效能，并为用户提供个性化和普适化的计算资源使用环境。虚拟化计算系统可以更加充分合理地利用计算资源，满足日益多样的计算需求，使人们能够透明、高效、可定制地使用计算资源，从而真正实现灵活构建、按需计算的理念。

为了满足不同的功能需求，目前已出现了许多不同种类的虚拟化解决方案，由于其采用不同的实现方式和抽象层次，使得这些虚拟化系统呈现出不同的特性。《计算系统虚拟化--原理与应用》[10]一书从虚拟机实现所采用的抽象层次的角度对虚拟化系统进行了如下分类：

1）**指令集虚拟化** 指令集虚拟化通过纯软件方法，模拟出与实际运行的应用程序（或操作系统）所不同的指令集去执行，采用这种方法构造的虚拟机一般称为模拟器（Emulator）。

2）**硬件级虚拟化** 硬件抽象层面（Hardware Abstraction Layer，即 HAL）虚拟化实际上与指令集架构级虚拟化非常相似，其不同之处在于，这种类型的虚拟化所考虑的是一种特殊情况：客户执行环境和主机具有相同的指令集合，并充分利用这一特点，让绝大多数客户指令在主机上直接执行，从而大大地提高了执行的速度。

3）**操作系统级虚拟化** 虚拟层按照每个虚拟机的要求为其生成一个运行在物理机器之上的操作系统副本，从而为每个虚拟机产生一个完整的隔离操作环境。

4）**编程语言级虚拟化** 在应用层次上创建一个和其他类型虚拟机行为方式类似的虚拟机，并支持一种新的自定义的指令集（例如 JVM 中的 Java 字节码）。

5）**程序库级虚拟化** 在操作系统层上实现了不同的应用程序二进制接口(ABI)和不同的应用程序编程接口(API)。

在这些不同层次的虚拟化技术中，硬件层次的虚拟化技术具有以下几个特性：可以将虚拟资源映射到物理资源，并在虚拟机计算中使用本地硬件。高度的隔离性（虚拟机和底层物理机器均实现隔离、易于让用户接受和用户所习惯使用的普通机器看起来一样）、支持不同操作系统和应用程序而不需要重启机器、低风险和易于维护等。对于硬件层次的虚拟化技术有着众多的解决方案，如 VMware Workstation[11]、Virtual PC[12]、Denali [13]、User Mode Linux[14][15]、KVM[16]和 Xen[17]等。Intel 和 AMD 都在这个层次上从硬件实现上对虚拟化进行了支持，而且现有的大部分商业虚拟化软件都使用此硬件辅助虚拟化技术来提高效率，这说明了此类虚拟化技术的可行性与实用性。本文研究中的虚拟化技术是指硬件级的虚拟化。

1.1.3 研究的提出

可信计算环境构建是通过软硬件结合的方式构建满足可信计算定义的系统，其目地是提升系统的安全性，关注内容是服务器、网络和终端的行为及其相互间的影响。构建可信计算环境需要可信计算平台及软件相互协作。可信计算平台是具有一定物理防护的计算平台，该平台可以提供一定级别的硬件安全来确保运行于该平台物理保护边界内的代码及数据具有某些特性，例如机密性、完整性、真实性等。可信计算平台中的可信硬件在其物理保护边界内的计算环境就是一个可信计算环境。物理保护代价是比较昂贵的，这也注定了单纯依赖硬件保护的可信计算平台其计算能力是有限的，不能满足大部分的应用需求，只能应用于一些特定的场景（例如用于保存个人密钥的个人令牌）。通常可信硬件都是做为可信计算平台附属设备用于增强整个平台的安全性。本文将单独依赖硬件保护的计算平台和将可信硬件作为附属设备的计算平台统一称为可信计算平台。

早期的可信计算平台大多是实验室的产品，近年来，随着 TCG 组织和以微软及 Intel 为代表的工业界的推动下，可信计算平台的商用化取得了长足的发展，特别是基于 TCG/TPM 规范[18]的可信硬件已经成为中高端商用 PC 和笔记本的标准配置。然而基于可信计算平台的应用却停留在非常小的领域之内。

阻碍可信计算走向应用的主要问题是传统软件架构很难为各种应用构建能够满足其安全需求的可信计算环境。传统软件架构以操作系统为中心，操作系统软件结构复杂、代码规模巨大，系统漏洞数量较多且很难发现，例如 UNIX 和 Windows 系统（包括标准应用）大约有 1 亿行代码。最新的研究表明[1]，每千行代码就有一个和安全相关的漏洞。即使在可信计算平台上将信任链扩展到应用程序这一层来保证从信任根到应用程序的真实性，操作系统的上述缺点也使其很难为上层软件提供足够的可信支持来确保其运行的安全性；同时运行于同一操作系统的多应用程序之间隔离性弱，即使相互之间没有调用关系也可能相互影响，那么要确保在操作系统中某个应用是可信性，就要确保运行于其上所有应用的可信性。这带来了实现上的难题，为了验证其可信性，系统需要维护数量巨大的可信软件的散列值，如果要将这样的验证方式扩展到存在着大量的异构计算平台，包括大型服务器、个人电脑和 PDA 等移动设备，并运行着不同的操作系统和应用软件的复杂网络环境中，计算平台的异构性将使验证数量巨大的可信软件散列值变得不可实现。

虚拟机架构能充分屏蔽下层硬件及指令集的差异，实现系统资源的虚拟化，从而充分适应应用软件对系统资源需求的多样性。虚拟机架构能够隔离软件与硬件、应用软件与底层系统之间的直接依赖关系。虚拟机架构的这些特点使其不仅仅用于资源聚合，还为解决传统软件架构在可信计算环境构建方面的不足带来了希望。基于虚拟机架构的可信计算环境可望替代基于操作系统的可信计算环境，实现系统的安全防护。同时虚拟化正在成为服务端应用的主流架构，也非常有必要对虚拟环境上应用软件行为进行有效地监管，为上层应用提供更强有力的保护和支撑。

1.2 国内外研究现状

1.2.1 可信计算环境构建的理论研究

现有的理论研究集中于以下几个方面：（1）在信任管理语境下对信任和信任度量模型的研究。1996 年，Blaze 等人[19]首先提出了信任管理的概念，Josang[20][21][22]提出了基于主观逻辑的信任模型（Subjective Logic），引入证据空间(Evidence Space)和观念空间（Opinion Space）的概念描述及度量信任关系。Beth 等人 [23]将信任分为直接信任和推荐信任，以对实体完成任务的期望为基础，根据肯定经验和否定经验计算出实体能够完成任务的概率，以此概率作为实体信任度的度量，并给出了信任推导和综合规则及相应的信任度计算方法。(2)在可信赖计算(Dependable Computing)语境下的可信理论研究。可信赖计算源自于容错，主要关注的是计算机软硬件系统从开发到使用整个生命周期中的可靠性、可用性、可生存性等问题[24][25][26]。（3）针对某些具体可信计算环境建模。Smith 基于 IBM 安全协处理器建立了对外认证(Outbound Authentication)模型[27]。该模型通过信任集和实体依赖函数等概念，给出了信任的形式化表述、对外认证机制的可靠性和完备性定义及证明。Abadi 等[27]利用安全逻辑语言对 NGSCB 系统中的鉴权和访问控制过程进行了形式化描述，该研究仅实现了对基于身份的鉴权过程的形式化分析，没有对可信计算模型的完整性进行验证。Chen 等[29]使用谓词逻辑对安全启动的过程进行了形式化的描述。Qu 等[30]对 Intel 的 Tboot 系统进行了形式化的建模。

1.2.2 可信硬件

可信计算平台是指能够通过一定程度的硬件安全来保证其上运行计算可信性的平台。可信硬件是可信计算平台核心组成部分。IBM 4758 安全协处理器 [31] 是早期的可信硬件，它是一个同主处理器相独立的处理器单元，用于负责和安全相关的计算。IBM 4758 设计目的是即使在存在本地敌手的情况下也能满足某些计算与存储性质。这些本地敌手可以是计算系统的操作者或是主处理器上正在运行的进程。安全协处理器可以提供一个隔离的运行环境确保该环境中计算的完整性和机密性，并能

对外证明在这个环境中进行运算的真实性。XOM[32]是斯坦福大学提出的一种基于只执行内存的安全增强型处理器结构。应用程序可以只信任处理器而不必信任操作系统。XOM 体系可加密进程执行空间，只有处理器处于可信模式下才能用密匙解密程序。AEGIS [33]是麻省理工学院提出一种安全增强性的处理器结构。类似 IBM 4758，该结构以处理器为信任根并假定处理器拥有一个秘密的密匙，AEGIS 架构的处理器新增一种类似 XOM 的“安全执行模式”，并增加了新的指令用于进入/退出“全执行模式”和使用受保护的秘密密匙。AEGIS 可以用于认证执行和数字版权保护。Ceriums[34]是麻省理工学院提出的另一种可信处理器。Cerium 结合上述增强型处理器的优点，它可以像 XOM 及 AEGIS 一样，通过加密被保护进程的地址空间来实现类似 IBM 4758 中的认证执行。然而不同的是 Gerium 采用软件的方式来实现加密保护进程， Gerium 将一个可信微内核放入处理器内部，对被保护进程地址空间的所有操作都会触发这段微内核代码，由它来处理加密地址空间。

主流的商用 CPU 中，Intel 和 AMD 分别推出了自己的增强型安全处理器技术--Intel 的 TXT[35]和 AMD 的 SVM[36]。不同于前面介绍的其他的增强型安全处理器，TXT 和 SVM 不提供加密进程空间的功能，仅在处理器中增加了新的指令作为动态可信度量根（Dynamic Root of Trust Measurement，DRTM），这条指令可以实现进程的保护执行，但是 TXT 或是 SVM 需要和 TPM（1.2 版本）相结合才能够提供封装存储和对外认证的功能。

基于 TCG 规范的 TPM 是商品化可信计算平台中的核心组件。TPM 提供一组平台配置寄存器（Platform Configuration Registers，PCR）进行平台的配置证明。拥有 TPM 的可信计算平台可以通过 PCR 来记录从系统加电启动后进行的操作顺序和参与的软件。主板 BIOS 中通常有一段称为可信度量根（Root of Trust for Measurement，RTM）的代码，随着平台的启动，通过一种“递归信任”的过程将信任由 RTM 扩展到整个平台。在“递归信任”过程中，前一阶段的代码在将控制权传递给下一阶段前，先对其进行度量，并把度量值保存到 PCR 中。TPM 提供可信报告根（Root of Trust for Reporting，RTR），RTR 通过签名来证实 PCR 中数据的真实性。TPM 还提供可

信的存储根（Root of Trust of Storage，RTS），用于保护所有委托给 TPM 的密钥和数据。

1.2.3 传统架构下的可信计算环境

拥有可信硬件的可信计算平台为构建可信计算环境提供了硬件基础，然而只有通过软件的协助才能构建满足多种应用需求的可信计算环境，这方面的研究在国外引起了广泛的关注。安全硬件加强的 MyProxy(SHEMP)[37]是 Dartmouth 大学 2004 年的一个项目，主要研究使用 IBM4758 协处理器来管理终端，通过控制安全边界来加强安全性，可用于 MyProxy 服务器。TrustedGRUB[38]是 Sourceforge 支持的开源项目，它扩展了原始的 GRUB 引导程序来支持 TPM 提供的递归信任。在 TrustedGRUB 的引导过程中，TrustedGRUB 扩展了引导加载程序 GRUB（Grand Unified Boot Loader）的 Stage1，可以度量 GRUB Stage2 的第 1 个扇区。包括 Stage2 的输入参数（内核，启动模块和相关配置信息）。TrustedGRUB 将信任链扩展到操作系统层。

BEAR[39][40]是 Dartmouth 大学 PKI 实验室的研究项目，主要研究在具有 TCG/TPM 的商用可信平台上使用 Linux 构建可信桌面计算环境，该可信计算环境需要修改 Linux 操作系统，但保持对操作系统上的应用透明，即传统软件和应用程序可以不加修改的在这个平台上运行。BEAR 将信任链扩展到文件层，操作系统下任意一个目录第一次被打开时，BEAR 都会检查该目录下文件的完整性。同时 BEAR 可以提供远程认证和安全存储功能。IMA[41]是 IBM 提出的一种可信计算环境构建架构，类似 BEAR 项目，IMA 同样是在拥有 TCG/TPM 的商用可信计算平台上使用 Linux 构建可信计算环境，并提供远程认证和安全存储的功能。不同是，IMA 采用在一些系统调用中加钩子的方式，当可加载内核模块和程序载入内存时，对其进行完整性校检，以使信任链可以扩展到应用程序层。

OSLO[42]（Open Secure Loader）是基于 AMD64 位处理器的动态可信度量根的开源引导加载器，它通过 AMD 动态可信度量根技术中新增加的 SKINIT 指令来替代基于 BIOS 的静态可信度量根。OSLO 启动加载器作为操作系统内核的一部分，它初始化 TPM，通过 TPM 设备驱动来实现与 TPM 通信，调用 SKINIT 指令，度量所有被

加载的文件，并将度量值扩展到 TPM 的动态 PCR 中。TBoot[43]（Trusted Boot）类似于 OSLO，但是 TBoot 使用的是 Intel TXT（Trusted eXecution Technology，简称 TXT）技术提供的动态可信度量根来度量操作系统内核/虚拟机监控器（Virtual Machine Monitor，简称 VMM）。TBoot 可以实现可信启动、撤销度量环境、重置数据保护、保护 TXT 内存范围等功能。Flicker[44]直接利用 AMD SVM 中新增的动态可信度量根扩展指令暂停操作系统和其上运行的其他程序，为敏感代码创建一个完全隔离于操作系统的运行环境，并可以实现远程证明和安全存储。

1.2.4 基于虚拟机架构的可信计算环境

在 20 世纪 70 年代，IBM 公司开发了安全内核的虚拟机监控模块 KVM/370[45]；在 20 世纪 80 年代，在 VAX 机上也开发了基于 VMM 的安全内核[46]；在 2000 年 4 月，IBM 又推出 z 系列大型安全服务器系统[47]。这些基于虚拟机的可信计算环境在大型机上的长期使用证明了基于虚拟机架构构建可信计算环境对增强系统安全的有效性。

斯坦福大学研发的 Terra 系统[48]是一个基于虚拟机架构的可信计算平台。Terra 通过可信的虚拟机监控器（TVMM）在具有防篡改功能（temper-resistant）的可信硬件平台上实现多个相互独立的虚拟机（VM）。这些虚拟机可以是具有通用性、对安全性没有特殊要求的 Open Box；也可以是有特殊用途的、高安全性的 Close Box。可信虚拟机的引入也为原有的专属平台上的应用，例如 ATM，提供了向普通商用平台移植的可能。IBM 实现的 VTPM[49]为单一硬件平台的多个虚拟机提供可信计算支持。通过虚拟化可信平台模块(TPM)，TPM 的安全存储和加解密功能能够应用于虚拟机上的操作系统和应用程序。原有的基于 TPM 的可信计算环境，在引入虚拟层后，可以将 TPM 的信任链向上扩展建立可信虚拟机，建立可信虚拟机的方式采用传统方式，如 IBM IMA 架构。Intel 在 2008 年提出了 VIS (Virtualization Enabled Integrity Services) 构架[76]，VIS 利用 Xen 虚拟机监控器的影子页表机制提出了一种多重影子页表的内存隔离机制，使得在虚拟机中运行的目标程序不被其他的进程篡改，并利用 TPM 来实现目标程序的完整性度量和敏感数据加密保护等功能。

基于虚拟机架构的可信计算环境中最具有代表性的是动态可信度量根技术同虚拟化技术相结合的 Intel TXT 安全架构[50]和微软的 NGSCB[52]。Intel TXT 安全架构是 Intel 在其增强型安全处理器上提出的一种可信计算环境构建架构。TXT 架构为被保护软件提供安全的执行环境，防止其他软件访问到被保护软件的资源。Intel TXT 并不像其他的增强型安全处理器一样具有安全模式，并当处理器离开安全模式时，由处理器来加密被保护软件的资源，将其同其他的软件实体隔离开。Intel TXT 的安全执行环境是通过迟启动（Late launch）技术暂停操作系统和其上正在运行的其他软件，并为安全执行环境提供动态可信度量根来实现的。如果要在 Intel TXT 的可信平台上同时运行不安全的代码与安全的代码，必须依赖软件的方法。虚拟化技术提供的隔离性正好可以满足这个需求，在支持 TXT 处理器中几乎都同时支持 Intel 的硬件辅助虚拟化技术。在 TXT 安全架构中除了要求平台的处理器支持 TXT 技术外，还要求平台拥有特定的芯片组，及可信的 I/O 外设（键盘、鼠标和显卡）和 TPM1.2。TXT 安全架构的设计目标提供以下的安全特性：

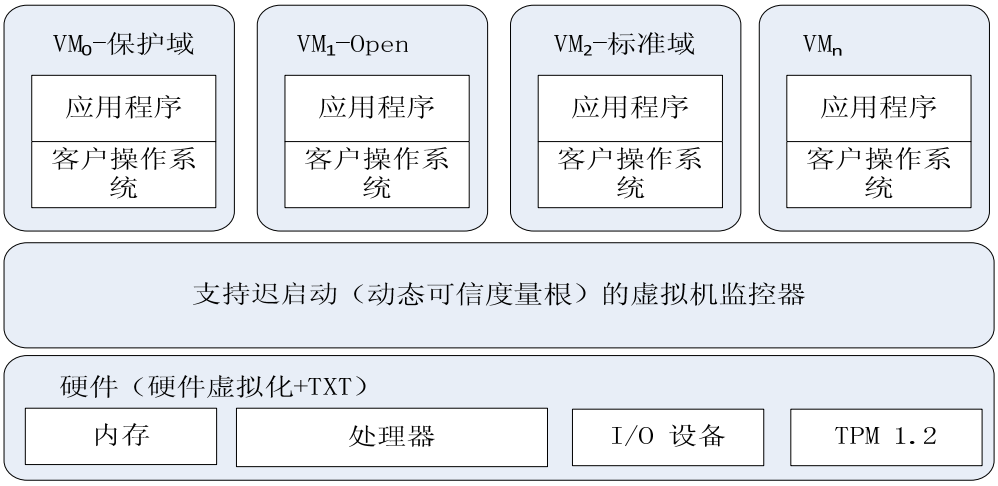

图 1.1 Intel TXT 安全架构

1）安全运行环境：为程序提供隔离的安全运行环境，防止其他未被授权的平台软件实体获取或是损害在这个保护运行环境中的程序。

2)封印存储：为密钥、数据和其他敏感内容提供基于平台可信硬件（依赖于 TPM）的封印存储--将这些数据同它们被存储时的系统环境绑定在一起，只有在同样的运行环境中才能被解密。

3）可信输入和显示：提供输入设备例如键盘/鼠标到安全运行环境中应用程序以及安全运行环境中应用程序到显卡缓冲区间的安全路径，防止平台上运行的其他没有被授权的软件实体窃取或是篡改输入和显示的信息（需要特定的可信键盘、鼠标和显卡）。

4）对外认证：提供安全执行环境和运行于其中程序的完整性证明。

图 1.1 展示 Intel TXT 安全架构。不同需求的操作环境被虚拟机监控器隔离到不同的虚拟域，这个架构同 Terrer 类似，但不同的是，依据 TXT 的设计虚拟机监控器可以依据保护需要动态加载而不需要重启系统，并通过 TPM1.2 可以提供从 TXT 动态可信度量根到虚拟机监控器的信任链证明。TXT 安全架构建议的隔离环境分为以下几种：

标准隔离域：该域提供一个标准的 IA-32 平台，在这个域中用户可以运行任何能在物理 IA- 32 PC 平台上运行的操作系统和软件。在 TXT 架构设计中，虚拟机监控器可以依据保护需要在运行的操作系统中启动，并将原有的操作系统和其上的应用软件迁移到标准的隔离域中。

保护域：该域同标准隔离域共同运行于虚拟机监控器之上。运行于该域的软件实体同标准域中的软件实体都是完全隔离的，不会被标准域的软件实体侵害。当保护域停止运行时，该域中的敏感数据将被保护到 TPM 中。

微软的下一代安全计算平台（Next-Generation Secure Computing Base，NGSCB）是 TXT 安全架构的一个特例化实现。虚拟机监控器将硬件资源分割成两个部分：左侧运行原生的遗留操作系统；右侧是一个微型的安全内核称为 nexus，其上运行的应用程序称为代理（agent），代理运行于一个隔离的地址空间并使用认证过的操作原语。虚拟机监控器和微安全内核一起为代理提供和 TXT 安全架构一致的安全特性：安全运行环境、封印存储、可信输入和显示及对外认证。

1.2.5 可信虚拟域

sHype[56]系统是由 IBM 提出并实现的，进而被 Xen 开源项目加入源码树中成为其可选的编译模块。sHype/ACM 为 Xen 虚拟机系统提供了一系列的安全特性，包括

安全服务（策略管理、编辑等）、资源控制、虚拟机之间的访问控制、虚拟资源的隔离以及基于 TPM 的证明机制等。sHype/ACM 通过加载不同的强制访问控制策略（目前支持中国墙策略[57]和 TE 简单类型策略[58]），能够帮助 Xen 对虚拟机的资源使用和虚拟机之间的信息流进行控制，同时借助基于 TPM 的证明机制能够实现对 Xen 和其上虚拟机的完整性度量，构建安全的可信虚拟域（Trusted Virtual Domains）。

Shamon[59]系统的设计基于 sHype/ACM 模块，遵守 Flask 架构[102]。在分布式环境下，Shamon 系统通过 TPM 实现各个节点的完整性度量和相互认证，从而将各个节点的参考监视器整合起来建立共享参考监视器（Shared Reference Monitor），即在节点间通过安全连接和相互认证使各个节点的参考监视器结合起来，形成一个概念上的单一参考监视器。共享参考监视器在整个分布式环境下对虚拟机的资源访问和通信执行强制访问控制，将 sHype/ACM 可信虚拟域由单节点扩展到分布式环境下。

IBM 将可信虚拟域 TVD 的概念[53][54][55]应用于数据中心等场景中，通过可信计算建立联合可信基（类似于 Shamon 的共享参考监视器），将不同节点的虚拟机依据负载的性质，分为不同的可信虚拟域。可信虚拟域内的多虚拟机之间是完全可信的，并通过定制的访问控制策略控制域间通信。

1.2.6 国内研究情况

2008 年 4 月底，中国可信计算联盟 (CTCU) 在国家信息中心成立。现已有 20 家正式成员，包含计算机厂商、信息安全厂商和一些应用厂商，以及国家的科研院所。 CTCU 的成立标志着中国在可信计算和信息安全领域的一次成功的尝试，标志着可信计算由理论逐步转化为产业，转入到实质性的实现阶段，并逐步开始应用到企业、政府、军事等领域。

在国内，对于可信计算平台的研究主要有：武汉瑞达公司和武汉大学合作生产的可信计算平台；联想集团的“恒智”芯片和可信计算机；兆日公司的 TPM 芯片。在理论方面研究主要有：沈昌祥等人对于计算平台的可信证明[62]和可信状态的安全模型[63]的研究及张焕国等人对可信计算平台信任链安全性[50]的分析。

华中科技大学和 HP、武汉大学、复旦大学合作开发了基于网格的可信计算平台 Daonity[64]，该项目通过引入可信计算技术改进现有的网格平台安全基础设施，项目针对 ChinaGrid 公共支撑平台 CGSP，使用可信平台硬件模块 TPM 和 TCG 软件栈 TSS 安全地产生、存储和使用加密证书，并利用 TPM 提高系统数据加密和文件保护的安全性，为网格使用者提供了一个更为可信的服务环境。

1.3 现有研究的不足

通过对可信计算研究历史的回顾和现状的分析，可以看到，基于虚拟机架构的可信计算环境构建机制是当前可信计算研究的热点，但是现有研究还存在着诸多的问题：

1）缺乏支持可信计算环境构建的理论模型。现有的信任和信任度量模型大多是在信任管理背景下进行研究的。信任管理适用于网络电子交易等应用场景中对访问控制和权限管理进行建模，而不适用于描述可信计算平台上的实体信任关系。而可信赖计算主要关注的是计算机软硬件系统从开发到使用整个生命周期中的可靠性、可用性、可生存性等问题。这些理论同可信计算(Trusted Computing)的研究内容区别较大，无法为可信计算环境建模。还有一些理论研究针对于具体的可信平台或是某一个特定的可信计算应用场景进行形式化的描述。这些理论不具备普遍意义，不能为构建可信计算环境提供理论上的指导，也不能作为评估现有可信计算环境的理论工具，这使基于虚拟机架构的可信计算环境构建缺乏理论上的支撑。

2）在基于虚拟机架构的可信启动方式上存在很大的缺陷。现有的研究将虚拟机监控器作为信任链中的一环，采用 TCG 的信任链扩展方式从可信根构建到应用层的信任链。这种方式需要修改操作系统的内核，使现有的操作系统无法支持应用程序级的完整性度量和认证。其次，该方式以操作系统为信任基（TCB），操作系统代码庞大，系统漏洞数量多且很难发现。第三，上述的方式只能为依赖方提供平台真实性的证明，而不能构建具有机密性保护能力的可信计算环境，难以满足分布式的应用。还有一些研究采用粗粒度的信任链构建方式，例如 Terra 以虚拟机镜像为单位进行度量。这种方式一方面会产生较大的度量值，以 4Gb 的镜像为例，Terra 大约要

产生一个 20Mb 的度量值。另一方面这种度量值不能用于解释度量内容，很难应用于远程认证的场景。

3）基于虚拟机架构的可信计算环境大多采用通用的虚拟机监控器。通用的虚拟机监控器主要是为服务器端的资源聚合而设计，其管理域因为资源管理和隔离等需求变得十分庞大（以 Xen 为例，不包括管理域，其 Hypervisor 的代码大约是 50,000 行）。这使通用虚拟机监控器作为可信基丧失了代码规模小，漏洞数量少的优势，并带来诸多的安全风险。同时虚拟化会带来系统开销，在用户不需要高安全性的环境时，却需要承受虚拟化带来的开销（特别是对于客户端的应用，大部分时间用户并不运行高安全性的应用）。 Intel TXT 和微软的 NGSCB 虽然提出了采用动态可信度量根来实现动态虚拟化的构想，然而其可信计算环境依然是建立在隔离域的基础上，这不可避免地使虚拟机监控器因为资源管理和提供有效的虚拟域隔离而变得庞大。

4）可信域间强制访问控制存在隐形流的风险，现有的可信机制不能解决这个问题。结合可信计算在虚拟机系统中实施强制访问控制策略构建的可信虚拟域，能够管理通过授权的公开通道传递的信息流，却不能够控制通过隐形通道传递的具有潜在风险的信息流。Trent Jaeger 等[60]虽然提出了可以通过加强访问控制策略的约束条件来控制隐形流，并讨论了使用现有访问控制模型，例如中国墙模型、TE 模型和 Caernarvon[61]等控制虚拟机系统中的隐形流。然而，使用现有的强制访问策略来处理隐形流，会使系统有太多的限制而缺乏灵活性和难以实施。如使用基于访问范围的 Caernarvon 模型需要事先划分通信集合，在一些分布式环境下，事先确定每一个通信集合是不可能的；基于流约束的中国墙策略则需要每个节点只运行单个虚拟机，过强的约束条件使中国墙策略没有实施的意义。同时，现有支持可信虚拟域的虚拟机系统也缺乏支持虚拟机域间隐形流控制的实施机制。

1.4 研究内容和意义

针对上述研究的不足，如图 1.2 所示，本文研究在不同应用场景中（服务端、客户端和分布式环境）如何构建虚拟机架构下具有真实性、完整性、机密性、可控性

等属性的可信计算环境，包括基于虚拟机架构可信计算环境构建的基本支撑理论（可信计算环境的信任链模型和可信虚拟域间隔离模型）和实现机制（动态信任链构建、内存隔离、轻量级虚拟化、透明信任链、完整性保护、联合可信基和访问控制实施等）。具体而言，主要工作和贡献集中在以下几个方面：

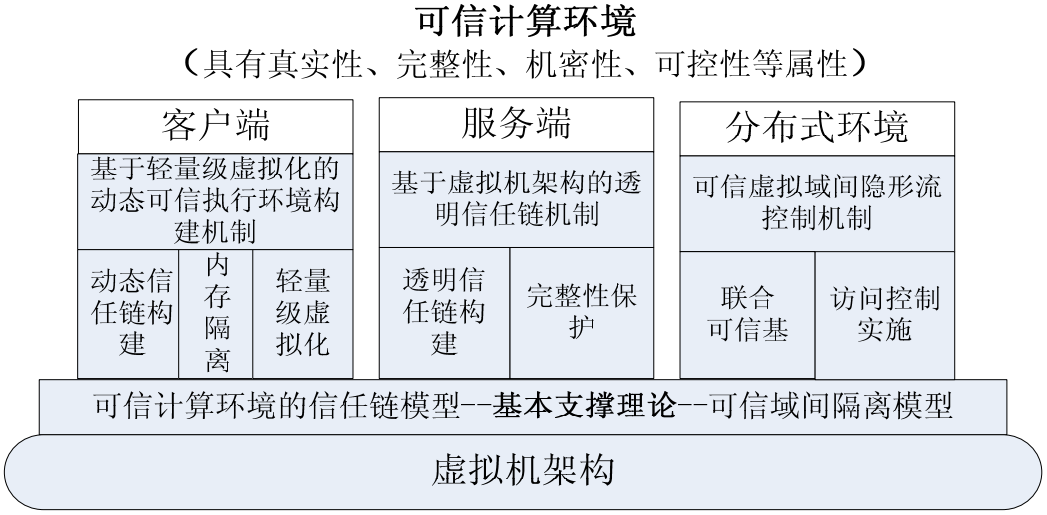

图 1.2 论文研究内容

1）基于虚拟机架构可信计算环境构建的支撑理论模型。本文提出可信计算环境的信任链模型和可信域间隔离模型。本文的信任链模型使用 Smith 的信任集和实体依赖函数概念，在真实性能够无损度量客体行为的假定前提下，给出可信状态、信任根、信任度量和信任链的形式化模型。该模型具有普遍意义，可以为评估现有的可信计算环境提供理论依据（包括基于静态可信度量根的可信计算环境和基于动态可信度量根的可信计算环境），并可以为判定可信计算环境信任链的正确性和构建更加合理的信任链提供理论支持。

本文提出可信虚拟域间隔离的优先中国墙模型，该模型通过建立联盟关系实现对利益冲突关系的动态扩展，从而构建访问区域。本文证明该模型能实现与 BN 中国墙模型相同的安全目标。优先中国墙模型具有和传统中国墙策略不同的访问规则，该模型能为用户提供一个可配置的调度方式来管理虚拟机资源从而控制可信虚拟域中的隐形流。优先中国墙模型既能保持类似中国墙策略在控制公开流时依据访问需要进行选择的特性，又具有类似格策略能够将客体分成不同访问集的特性。同时，

该模型不但能用于可信虚拟域中进行隐形流控制，还可以用于分布式联盟系统中来建立更加灵活的访问控制区域。

2）在可信计算环境的信任链的理论支撑下，本文提出基于虚拟机架构的透明信任链机制—TCT（Transparent Chain of Trust），TCT 能够构建对操作系统透明的应用程序级信任链，并可以在用户期望的计算环境遭到破坏时，保护其指定敏感数据。TCT 对操作系统和其上层应用的透明性，解决了虚拟机架构下信任链过长的问题。同时 TCT 可以提升普通商业平台的安全性，使其具有类似 IBM 4758 协处理器平台的完整性和机密性特性。TCT 可以用于虚拟机架构的网格和云计算等多重独立软件授权（Authorities）环境中，保护上层用户的数据隐私及系统的完整性。

3）基于本文的信任链理论，提出基于轻量级虚拟化的动态可信执行环境构建机制—Cherub。Cherub 构造从动态可信度量根到轻量级虚拟机监控器的动态信任链--在运行的操作系统下插入可信的轻量级虚拟机监控器，同时，轻量级虚拟机监控器使用双重影子页表机制为目标程序选择的内存页提供隔离的运行空间，从而将动态可信度量根到轻量级虚拟机监控器的信任链扩展到目标程序。

同 TCT 架构的目的不同，Cherub 主要用于提升桌面系统的安全性。同现有的研究相比，Cherub 有着诸多特性： Cherub 是基于动态可信度量根技术的首个虚拟化实现，Cherub 的虚拟机监控器可以在操作系统运行时动态地、可信地创建和撤销。因此，在系统不需要运行高安全的应用时，不必承受虚拟化的开销；Cherub 在原操作系统内部为目标程序隔离安全的执行环境，不需要创建新的虚拟域，因此其代码量非常小。同通用的虚拟机监控器相比，它拥有更小的可信基及启动和运行开销；Cherub 不需要修改操作系统和现有的编程模型，因此它可以更大范围地保护运行在现有商用操作系统上的遗留程序，这是基于半虚拟机架构和其他安全协处理的安全方案所不能解决的； Cherub 可以通过细粒度的内存访问控制策略灵活地保护全部程序或是部分程序。

4）本文提出一种基于优先中国墙策略的可信虚拟域间隐形流控制机制- CFC（Covert Flows Confinement）。CFC 构建联合可信基，并将访问控制的策略和实现机制进行分离，可以弥补现有支持强制访问控制的可信虚拟域架构的不足，控制可

信虚拟域间的隐形流问题。该机制对虚拟机透明，能够兼容现有的分布式虚拟机管理系统，并支持可信虚拟域内的多种访问控制策略。CFC 能满足虚拟机架构中企业级的高安全需要，确保其敏感的信息不会泄露给竞争对手。

综上所述，本文在信任链和可信虚拟域间隔离方面进行理论研究，并对虚拟机架构下透明信任链构建、完整性保护、程序可信执行环境和隐形流控制等方面给出有效的解决方案，对于构建基于虚拟机架构的可信计算环境来解决计算机安全面临的挑战具有重要的理论和应用价值。

1.5 论文组织结构

本文主要开展基于虚拟机架构的可信环境构建机制研究，内容组织结构如图 1.3 所示。

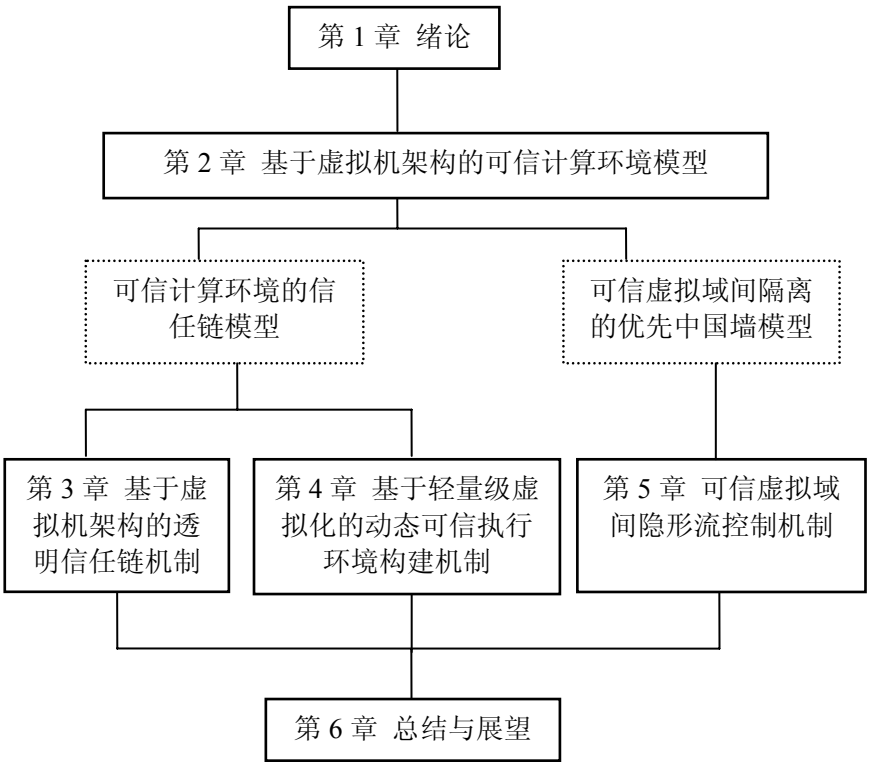

图 1.3 论文组织结构

第一章为绪论。本章介绍了基于虚拟机架构的可信计算环境构建研究背景，然后分析了当前国内外研究现状及不足，接下来指出了本文的研究内容和贡献，最后介绍本文的组织结构。

第二章是后续各章研究的理论基础，该章首先通过实体间的依赖关系等概念给出可信的形式化定义，然后给出可信度量、信任根的形式化模型，并对基于静态可信度量根和动态可信度量根的信任链进行统一的建模，同时使用该模型评估现有的可信计算环境。接下来，分析可信虚拟域间隐形流控制的需求，提出可信虚拟域间隔离的优先中国墙模型，并给出新访问规则的形式化描述和控制实例，最后证明该模型能够满足 BN 中国墙策略的安全目标，并分析该模型与 BN 和 Volker 模型相比较在可信虚拟域间隐形流控制及分布式联盟等场景中的优势。

第三章根据本文提出可信计算环境的信任链模型，首先给出基于虚拟机架构的透明信任链机制—TCT 的真实性证明和机密性保护需求的形式化描述，然后介绍 TCT 架构的设计和逆向语义解析、磁盘内存访问控制及系统调用追踪等实现技术，最后描述 TCT 在基于虚拟机架构的云计算环境中的应用并分析 TCT 对系统性能的影响。

第四章首先给出基于轻量级虚拟化的动态可信执行环境构建机制--Cherub 的威胁模型、设计目标和面临的挑战，然后给出轻量级虚拟机和保护执行环境的设计及实现，接下来对 Cherub 安全性进行分析，最后从 Cherub 的启动和运行等多个角度评估 Cherub 机制对性能的影响。

第五章首先分析可信虚拟域间隐形流访问控制机制--CFC 的设计需求，然后给出优先中国墙策略的实施算法，接下来给出策略管理器、安全监控器的架构设计与实现技术，最后分析 CFC 机制对性能的影响。

第六章总结全文，并展望基于虚拟机架构的可信计算环境构建机制研究的可能趋势和未来工作。

最后是参考文献和致谢。

附录 1 为作者在攻读博士学位期间发表的学术论文，附录 2 为申请的发明专利和软件著作版权，附录 3 为作者在攻读博士学位期间参与的科研项目，附录 4 为作者的个人简历。

2 基于虚拟机架构的可信计算环境模型

在以可信计算组织（TCG 规范）、微软（NGSCB）和 Intel（TXT）及 AMD（SVM）等研究机构和工业界的推动下，可信硬件平台已经商用化，然而在上述平台上的应用却局限在狭小的领域内。缺乏理论上的指导是影响可信计算应用发展的障碍之一，也使基于虚拟机架构的可信计算环境构建缺乏理论上的支撑。为了解决这个问题，本章给出基于虚拟机架构的可信计算环境构建的基本支撑理论—可信计算环境的信任链模型及可信虚拟域间隔离模型，本章模型是本文后续研究的理论基础，分别给出了不同应用场景（客户端、服务端和分布式环境中）中构建虚拟机架构下可信计算环境的支撑理论。

2.1 研究背景

目前可信计算领域存在的主要问题是理论研究滞后于可信计算技术的发展，现有的理论研究集中于信任管理、可信赖计算（Dependable Computing）或是针对具体的可信平台或是某一个特定可信计算应用场景进行形式化的描述。在信任管理或是可信赖计算背景下进行的理论研究同可信计算(Trusted Computing)的研究内容区别较大，无法为可信计算环境建模。而对于具体的可信平台或是某一个特定可信计算应用场景进行的形式化描述，不具备普遍意义，不能为构建可信计算环境提供理论上的指导，也不能作为评估现有可信计算环境的理论工具。同时结合可信计算在虚拟机系统中实施强制访问控制策略构建的可信虚拟域，能够管理通过授权的公开通道传递的信息流，却不能够控制通过隐形通道传递的具有潜在风险的信息流。现有的控制模型不能满足可信虚拟域间的隐形流控制需求。

在前人研究中，与本章信任链模型最为接近的是 Smith 的对外认证(Outbound Authentication)模型[27]。该理论模型通过建立信任集和实体依赖函数等概念，给出信任的形式化表述和对外认证机制的可靠性及完备性定义和证明。但 Smith 的对外认证理论模型有明显的局限性，它基于 IBM 4758 安全协处理器进行建模，IBM 4758

安全协处理器中可信计算环境是在其物理保护边界之内的，而现有的可信平台（TCG/TPM 及 Intel 和 AMD 的动态可信度量根）需要构建的可信计算环境超越硬件的物理保护边界，因此 Smith 的模型不适宜描述现有可信计算平台的可信环境构建。

依据可信计算平台的主要实现方式，可信计算环境可以分为两类：一类是物理安全保护边界内的可信计算环境（如在 IBM4758 平台上的计算环境和智能卡上的各种应用）；另一类通过软件将可信计算环境扩展到物理安全保护边界之外（如基于 TCG 规范及 Intel 或 AMD 动态可信度量根的各种可信计算环境）。这两种分类本质上只是在可信计算环境的构建上对平台实体的功能进行了不同的软硬件分工，这种分工取决于安全防护等级和成本上的平衡。任何可信计算环境都不可能由单一的实体（模块）构成，而是由多个相互影响的实体构成。本文提出的模型关注的是一个可信计算环境的外部实体或该计算环境用户如何判定该可信计算环境中的某个实体或是整个计算环境是值得信赖的，以及构成可信计算环境的实体功能如何分解和如何传达平台实体的可信任（Trustable）信息。基于特定平台的 Smith 模型不具备普遍性，但其在信任集和实体依赖函数概念下给出的信任描述能够满足 TCG 可信计算语境下建模的需求。本章将沿用 Smith 对信任的定义给出一个可信计算环境的信任链模型。

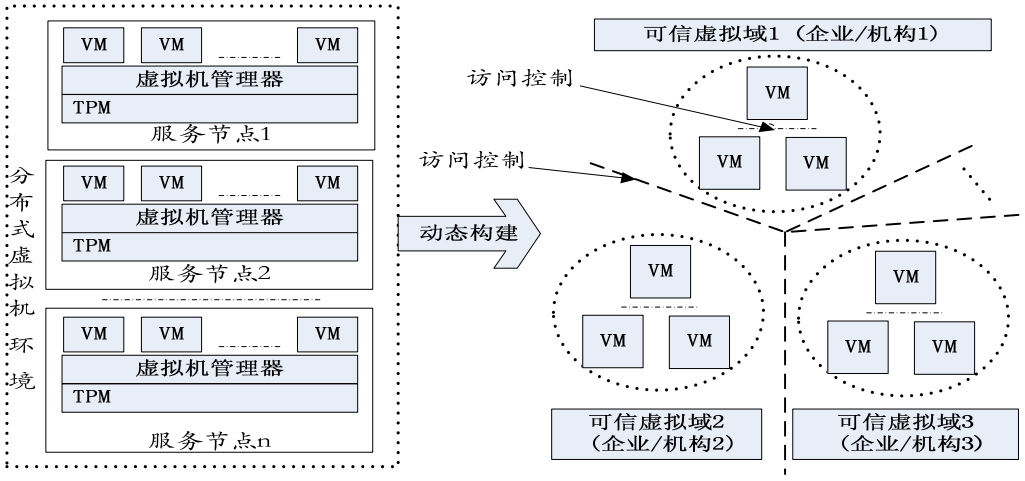

图 2.1 可信虚拟域

在虚拟机系统中，多个虚拟机间共享资源、进行信息传递是很普遍的要求，但是这种资源共享和信息传递可能会导致虚拟机间的信息泄露。结合可信计算和强制

访问控制，在虚拟机系统中构建可信虚拟域（如图 2.1 所示）可以通过增强虚拟域间的隔离性并提供完整性保护来提升虚拟机系统的安全性和可管理性。

IBM 提出了可信虚拟域 TVD[53]-[56] （Trusted Virtual Domains）的概念，可信虚拟域直接将可信计算技术整合到虚拟机系统管理软件中，并依据负载的不同将虚拟机和其他资源组织成不同的虚拟域，在虚拟域间实施强制访问控制。TVD 使用 TCG 的可信启动度量平台上所有运行软件的完整性和进行身份认证，并构建跨平台的分布式联合可信基。联合可信基在整个分布式环境下对虚拟机的资源访问和通信执行强制访问控制，并通过强制访问控制策略定义了虚拟机可以访问的资源，可以进行通信的虚拟机集合，从而构建跨节点的分布式可信计算环境（可信虚拟域）。

在可信虚拟域间，两个虚拟机（VM）虽然被禁止公开地信息传递，但仍可能通过隐形通道传递信息。1983 年美国国防部在其发布的可信计算机系统评估准则(TCSEC)[81]中，明确提出隐通道的问题，并规定在 B2 级及以上的高等级可信系统设计和开发过程中，必须进行隐通道分析。我国国标 GB／T18336[82]、国际标准 ISO/IEC 15408[2]等均把隐通道分析的强度作为评估一个高等级可信系统的硬性指标。实施强制访问控制的可信虚拟域间能够管理通过授权的公开通道传递的信息流，却不能够控制通过隐形通道传递的信息流。当虚拟化逐渐成为数据中心、云计算等服务端应用的基础架构时，这种威胁已经成为迫切要解决的安全问题，因此在可信虚拟域间必须处理隐通道带来的信息泄露风险。本章将分析可信虚拟域间的隐形流控制需求，并给出一个可信虚拟域间隔离模型。

2.2 可信计算环境的信任链模型

2.2.1 可信状态

假设 计算环境只有一个内存区域可以驻留软件并且驻留在该区域的软件不会受到计算环境外部的篡改。当该计算平台重新加电启动时，其内存区域的所有状态清零。本文将满足上述假设的计算环境记为 $\&_{CE}$。

上述对计算环境 $\&_{CE}$ 的假设并不局限于特定的可信计算平台，从早期基于 IBM4758 协处理器的可信计算平台到当前的基于 TCG/TPM 及 Intel 或 AMD 动态可信度量根的可信计算平台都能满足上述假设的条件。

定义 2.1. 时刻 $T=\{0,1,...,t,...\}$ 定义了一个时间序列。

定义 2.2. 程序集 $P=\{p_1,p_2,...,p_n\}$ 是 $\&_{CE}$ 中所有具有计算能力的代码和其相关结构数据的集合。

定义 2.3. 实体是程序 p 在一个特定时刻被载入 $\&_{CE}$ 中并执行。实体集 E 可以表示成 $\&_{CE}$ 中 $P\times T$ 上的一个二元关系，记为 $E=\{(p_1,t_1),(p_2,t_2),...,(p_n,t_n)\}$。

上述定义的实体，并不等同于程序，而是与程序和程序载入的时刻相关。相同的程序在不同的 $\&_{CE}$ 中载入被认为是不同的实体，相同的程序在同一 $\&_{CE}$ 中的多次载入也是不同的实体。

定义 2.4. t 时刻，系统的状态 S_t 可以被表示为 t 时刻在 $\&_{CE}$ 中实体的集合，即 $S_t\in 2^E$。

在一个时刻 t，系统的状态 S_t 不仅决定于 $\&_{CE}$ 中正在运行所有实体，还决定于在这个时刻以前实体的动作序列。当程序 p 在时刻 t 被载入 $\&_{CE}$，系统状态将发生相应的变化。在 t 时刻系统状态 S_t 是由 t-1 时刻的系统状态 S_{t-1} 和在这一刻被载入的程序 p 所决定，即 $S_t-S_{t-1}=\{(p,t-1)\}$。上述定义中，本文假定程序的载入是时刻 t 的一个原子操作，即在时刻 t 只有一个程序被载入。

实体之间相互作用，它们之间的作用关系不同，一类实体对另一类实体有读写或是控制的能力，因此后者的行为正确性依赖于前者。

定义 2.5. 依赖关系[27]：对 $\forall e_1,e_2\in E$， 如果 e_1 能够读/写 e_2 的数据，则称 e_2 数据依赖于 e_1，记为 $e_2Dep_{data}(e_1)$；如果 e_1 可以写/控制 e_2 的代码，则称 e_2 控制依赖于 e_1，记为 $e_2Dep_{code}(e_1)$。

推论 2.1. 基于定义 2.5，有以下推论：

$$\begin{cases} e_1 \in Dep(e_1) & \text{幂等性} \\ \text{如果 } e_2 \in Dep(e_1), \text{那么 } Dep(e_2) \in Dep(e_1) & \text{传递性} \end{cases}$$

令 $\xrightarrow{t}$ 表示在 S_t 中 Dep_{data} 和 Dep_{code} 两种关系并集的传递闭包。$Dep_t(e)=\{f : e \xrightarrow{t} f\}$ 列出了 t 时刻在 $\&_{CE}$ 中能够影响 e 操作正确性的所有实体。

定义 2.6. 依赖方 A 是计算环境 $\&_{CE}$ 外的一个实体或是平台用户，需要确认计算环境 $\&_{CE}$ 内实体 e 或是计算环境 $\&_{CE}$ 中的某个系统状态 S_t 是否可信。

平台实体 e 是否可信，取决于依赖方的判断，不同的依赖方对平台中的那些实体是可信的通常有着不同的看法，并与依赖方的应用相关。

定义 2.7. 对于依赖方 A，可信集代表某些实体的集合，A 相信这些实体的行为是可信的。本文将可信集记为 $Trustset(A)$。

定理 2.1. 在时刻 t，对依赖方 A，平台 $\&_{CE}$ 上有实体 $e=(p,j), j<t$，如果 $Dep_t(e) \subseteq Trustset(A)$ 那么实体 e 在时刻 t 对于依赖方 A 是可信的。

证明：由推论 2.1 可知，在时刻 t，平台 $\&_{CE}$ 上实体 e 的行为正确性取决于 $Dep_t(e)$，因此对依赖方 A，e 是否可信取决于 $Dep_t(e)$ 中所有实体行为对依赖方 A 是否可信的。故当 $Dep_t(e) \subseteq Trustset(A)$ 时，由定义 2.7 可证。

推论 2.2. 由定理 2.1，假设平台 $\&_{CE}$ 上有实体 e 对依赖方 A 在时刻 $t\text{-}1$ 是可信的，那么在时刻 t，如果有 $Dep_t(e)=Dep_{t-1}(e)$，则实体 e 在时刻 t 对于依赖方 A 也是可信的。

在时刻 t，系统 $S_t \in 2^E$，因此由定理 2.1 可以得到 S_t 可信的判定：

推论 2.3. 由定理 2.1，对依赖方 A，在时刻 t，系统状态 S_t 是可信的当且仅当 $\forall e \in S_t, Dep_t(e) \subseteq Trustset(A)$。

2.2.2 信任根和信任度量

定义 2.8. 对依赖方 A，可信证明集是实体行为证据的集合，A 信任这些证据关于实体行为的陈述并认为这些实体的行为是可信的。本文用 Trusted（A）来表示依赖方 A 的可信证明集。

定义 2.9. 设 M 是 $\&_{CE}$ 中所有实体行为证据的集合，信任度量是一个映射 $v \in V : E \times E \to M$ 。

定理 2.2. 在时刻 t，如果平台 $\&_{CE}$ 上的实体

$e = (p, j), j < t, Dep_t(e) - \{e\} \subseteq Trustset(A) \wedge \exists e' \in S_{j-1}$，

$Dep_{j-1}(e') \subseteq Trustset(A) \wedge v(e', e) \subset Trusted(A)$ 则对依赖方 A，e 是可信的。

证明：在时刻 t，实体 $e = (p, j), j < t$

由条件 $\exists e' \in S_{j-1}, Dep_{j-1}(e') \subseteq Trustset(A) \wedge v(e', e) \in Trusted(A)$ 和定义 2.8 及定义 2.9 可知 $e \in Trustset(A)$

同时因为 $Dep_t(e) - \{e\} \subseteq Trustset(A)$

可得 $Dep_t(e) \subseteq Trustset(A)$

由定理 2.1 可证。

推论 2.5. 由定理 2.2，如果在时刻 *t-1*，对依赖方 *A*，平台 $\&_{CE}$ 上的实体 *e* 是可信的，在时刻 *t* 如果 $S_t - S_{t-1} = \{e''\} \wedge \exists e' \in S_{t-1}$， $Dep_{t-1}(e') \subseteq Trustset(A) \wedge v(e', e'') \subset Trusted(A)$，则在时刻 t，e 对依赖方 *A* 是可信的。

定义 2.10. 在平台 $\&_{CE}$ 上，如果对任意依赖方 *A* 和时刻 t，

有 $e' = (Root, t-1), Dep_t(e') = \{e'\} \wedge e' \in Trustset(A)$，

则程序 $Root \in P$ 称为平台的信任根，实体 e' 称为平台的信任根实体。

在平台 $\&_{CE}$ 上，信任根实体是一个特殊的实体，它是平台的信任根 Root 在某一个时刻被加载形成的实体。对该平台的任意依赖方，它都是无条件可信的，并且它的可信性不依赖于平台其他的实体。

定理 2.3. 对依赖方 *A*，Root 是平台 $\&_{CE}$ 的信任根，如果系统状态转换满足以下两个条件：

$$\begin{cases} S_1 = \{(Root, 0)\} & (1) \\ 当 1 < k < t, S_{k+1} - S_k = \{e_k\} \vee \exists e' \in S_k, v(e', e_k) \in trusted(A) & (2) \end{cases}$$

在时刻 t，S_t 是可信的：

证明：对系统状态序列$S_1,S_2,...,S_{t-1},S_t$；

N=1 时 系统状态 $S_1=\{(Root,0)\}$

因为 Root 是平台信任根，由定义 2.10 可知$Dep_1(e')=\{e'\}\wedge e'\in Trustset(A)$

因此S_1的状态是可信的；

N=k 时，假设S_k是可信的；

对于 N=k+1，$S_{k+1}-S_k=\{e_k\}$，

因为S_k是可信的，由条件（2）和推论 2.5 可知$\forall e\in S_k, e\in Trustedset(A)$；

由推论 2.4 可知状态 S_{k+1}是可信的。

综上所述结论得证。

2.2.3 信任链模型

定义 2.11. 在平台$\&_{CE}$上，对依赖方 A，如果在时刻 t，$\exists E'=\{(Root,t_0),(p_1,t_1),...,(p_k,t_k)\}\subseteq S_t, 0\le t_0<t_1<,...,<t_k<t, \ \forall t_k<j<t\in T$：

$Dep_j((p_k,t_k))-\{(p_k,t_k)\}=Dep_j((p_{k-1},t_{k-1}))\wedge$

$Dep_j((p_{k-1},t_{k-1}))-\{(p_{k-1},t_{k-1})\}=Dep_j((p_{k-2},t_{k-2}))\wedge \ ...\wedge Dep_j((Root,t_0))=\{(Root,t_0)\}\wedge$
$\forall(p',t')\in E'-\{(Root,t_0)\},\exists t''<t',(p'',t'')\in E': v((p'',t''),(p',t'))\in Trusted(A)$
则称从$(Root,t_0)$到$e=(p_k,t_k)$在时刻 t 存在信任链。

定理 2.4.对依赖方 A，假设 $p_n\in P, 0\le t_0<t_n<t$，如果从$(Root,t_0)$到$e=(p_n,t_n)$在时刻 t 存在信任链，则 e 在时刻 t 对依赖方 A 是可信的。

证明：

因为由定义 2.11 可知如果$(Root,t_0)$到$e=(p_n,t_n)$在时刻 t 存在信任链，

则$\exists 0\le t_0<t_1<,...,<t_n<t, E'=\{Root,t_0),(p_1,t_1),...,(p_n,t_n)\}\subseteq S_t\wedge\forall t_0<j<t,$

$Dep_j(p_k,t_k)-\{(p_k,t_k)\}=Dep_j(p_{k-1},t_{k-1})\wedge$

$Dep_j(p_{k-1},t_{k-1})-\{(p_{k-1},t_{k-1})\}=Dep_j(p_{k-2},t_{k-2})\wedge ...\wedge Dep_j(Root,t_0)=\{(Root,t_0)\}$，

所以 $Dep_t(p_n,t_n)\subseteq\{(Root,t_0),(p_1,t_1),...(p_n,t_n)\}$；

同时$\forall(p',t')\in E'-\{(Root,t_0)\},\exists t''<t',(p'',t'')\in E': v((p'',t''),(p',t'))\in Trusted(A)$；

由定理 2.2 可知：$Dep(p_n,t_n)\subseteq\{(Root,t_0),(p_1,t_1),\ldots,(p_n,t_n)\}\subseteq Trustse(A)$，

由定理 2.1 可证。

2.3 可信计算环境的信任链分析

定理 2.3 给出一种信任链构造方式（假定选用的度量方式可以无损度量实体的行为）。TCG 的可信启动方式是定理 2.3 的一个特例，定理 2.3 可以用于证明 TCG 可信启动过程中信任链构建的正确性。TCG 的可信启动是从信任根开始，一级测量认证一级，一级信任一级，把信任关系扩大到整个计算机系统。TCG 可信启动过程中信任链构建方式限定了定理 2.3 条件 (2) 当 $1<k<t,\quad S_{k+1}-S_k=\{e_k\}\vee\exists e'\in S_k, v(e',e_k)\in trusted(A)$ 中 e' 的选择，要求 e' 是 *k-1* 时刻载入的实体。

定理 2.3 同样为前言中介绍的 Dartmouth 大学的 BEAR[39][40]和 IBM 的 IMA[41]系统的信任链设计提供了正确性证明。这两个系统的信任链构建首先依照 TCG 的可信启动方式扩展到操作系统层，然后由操作系统度量后续的实体。这些后续的实体依据实际的需求载入，而不是类似于 TCG 可信启动中规定的采用固定序列载入。定理 2.3 的条件（2）并不要求固定载入序列，因此上述系统的信任链构建方式能够满足定理 2.3 的条件，故可知其信任链设计是正确的。

从定理 2.3 中可以看出，构造基于静态信任根的信任链需要保证从系统加电载入可信根开始到验证时刻 t，所有载入平台 $\&_{CE}$ 内存中的实体都是可信的。例如，即使忽略构造静态信任链在度量过程中的性能问题，构建一个基于 Windows XP 的可信环境，就必须度量 Windows XP 本身及能在其上运行的成千上万的程序，这需要维护数量巨大的可信软件的散列值。在实现上，构建这种静态信任链几乎是不可能的。

2.2 节中的信任链模型对基于动态可信度量根的信任链和基于静态可信度量根的信任链进行了统一的建模。定理 2.3 可以看做是定理 2.4 一个特殊情况，即信任根在系统加电时载入，并且从 $(Root,t_0)$ 到 $e=(p_n,t_n)$，在时刻 t 存在一个包括所有被加载实体的信任链。依赖方 A 可以只关心同其交互的实体 e 是否可信，而并不需要平台 $\&_{CE}$

上整个系统S_t是可信的。因为从定义 2.11 可以得出动态信任链的构建方式：即从平台实体中选取一个包含信任根的子集并在这个子集上构建信任链。动态可信度量根可以在$0 \le t_0 < t_n < t$的任意时刻加载，并不要求其从$t_0 = 0$开始，但必须保证所选子集中的实体不依赖于其他平台实体。

定理 2.4 可以用于判定基于动态可信度量根的可信计算环境中信任链的正确性。以 Flicker 系统为例，该系统利用 AMD SVM 中新增的动态度量根扩展指令提供的隔离性来执行一段代码。在 Flicker 系统中，如果假定动态可信度量根实体$(Root, t_0)$在t_0时刻加载，被执行的这段代码对应的实体(p, t_1)则在t_1时刻加载。在实体(p, t_1)和动态可信度量根实体$(Root, t_0)$之间，没有其他实体加载。在 Flicker 系统中，动态可信度量根指令可以通过暂停原有系统的执行确保实体(p, t_1)不依赖于t_0时刻前的实体，并而在操作系统恢复时刻t以前清除实体(p, t_1)，从而保证实体(p, t_1)不依赖于t后的实体。由定义 2.11 和定理 2.4 可知 Flicker 系统的信任链设计是正确的。

Flicker 的安全执行环境是以暂停操作系统和其上正在运行的其他软件来实现的。因此如果要在动态可信度量根加载后，同时在平台上运行不安全的代码与安全的代码，依照定理 2.4，必须保证可信代码同不可信代码之间的隔离性，这需要依赖软件的方法。虚拟化技术提供的隔离性正好可以满足这个需求，在支持动态可信度量根的处理器中几乎都同时支持硬件辅助虚拟化技术。Intel TXT 和 NGSCB 的设计都符合定义 2.11 的要求，动态可信度量根可以保证信任根实体自身和虚拟机监控器从可信根加载t_0时刻到操作系统恢复t_n的隔离性，从而使上述实体不依赖于平台在时刻t_0前加载的实体。运行态 VMM 及其可信域的隔离性则由虚拟化技术提供，使这些模块实体在t_n时刻后不依赖于操作系统和其他平台实体，从而构建出符合定义 2.11 的动态信任链。定义 2.11 和定理 2.4 也为本文后续研究构建更加合理的信任链提供了理论的支持。

2.4 可信虚拟域间隐形流控制需求

2.4.1 隐通道

隐通道的存在为高安全需求的系统带来了潜在的风险。隐通道以一种违背系统安全策略的方式传递信息，包括存储隐通道与时间隐通道。如果传递信息的双方通过直接或间接地读写一存储位置来进行通信，这个隐通道是存储隐通道。如果传递者通过调整的系统资源(如 CPU)的使用时间影响接收者的实际响应时间来传递信息，这个隐通道是时间隐通道。例如，在传统操作系统中，磁盘移臂隐通道可以看作是存储隐通道，因为发送进程能够改变磁臂的运行，接收进程能够观察到这种变化。而通过对 CPU 的使用来传递信息的方式可以看作是一个时间隐通道，因为接收进程可以通过一定的时间间隔对发送进程的 CPU 使用情况进行测量来接收信息。

引起隐通道的原因有两种：一是系统设计中的漏洞，二是因为资源共享。国内外提出了很多隐通道的消除方法，这些方法主要是通过重新修改系统软件的细粒度资源控制代码，较常见的有泵协议法[83]、存储转发法[84]、混沌时间法[85]、向上通道[86]等。隐形通道很难鉴定，有些隐通道根本无法消除；而且，消除隐通道往往需要改变系统的设计与实现，例如，消除隐通道可以利用的共享资源；修改隐通道可以利用的接口约定等。因此，有些隐通道尽管可以消除，但消除代价太大。

2.4.2 中国墙模型

中国墙模型的构想来源于证券交易，证券顾问或市场分析师为证券交易的单位做业务咨询，他必须对单位信息保密，而且不允许用内部信息为相关的利益竞争者服务。

1989 年，Brewer 和 Nash 根据现实的商业政策模型定义了中国墙模型，也称 BN 中国墙模型[57]，它可以防止由于信息在竞争对手间流动带来的利益冲突，因而常用于商业环境中。中国墙模型根据主体已经拥有的访问权利来确定是否可以访问当前数据。中国墙模型结合了自主访问控制与强制访问控制的特征，具有如下性质。

1) BN 简单安全规则，一个主体 s 可以读数据对象 *o* 的条件是满足下列两者之一：① 对象 *o* 与主体 *s* 已经访问过的对象属于同一公司的数据集；② 对于主体 *s* 访问过

的每一个对象，其所属公司数据集与对象 *o* 所属的公司数据集不属于同一利益冲突类；2) BN*规则，主体 *s* 写数据对象 *o* 需要同时满足如下 2 个条件：① 据 BN 简单安全性质，主体 *s* 可以访问该对象 *o*；② 对于所有主体 *s* 访问过的对象，必须与主体 *s* 将要写的对象 *o* 在同一个公司数据集中。

对中国墙策略的研究集中于以下几个方面：Lin[87]详细分析了中国墙模型冲突关系的数学定义，针对利益冲突不具有传递性，对中国墙模型进行扩展。Meadows 等[88]对中国墙模型与 Bell-LaPadula 策略做了深入的对比研究。Sobel[89]利用迹(trace)方法分析研究中国墙安全策略，提出了具有时间间隔和访问权利撤消的利益冲突处理的新思想。Lin[90]则利用有权二元关系(或模糊二元关系)提出了另一种分析利益冲突的方法。Volker[91]提出一种中国墙的变型，该变型部分的动态扩展了冲突关系，扩大写权限的范围。在模型应用实施方面 Sandhu[92]提出用基于格(1attice)的访问控制模型，RBAC 来实施中国墙模型，何永忠[93]等提出了配置 RBAC 实施自主访问控制策略和强制访问控制策略的完整方案。秦超[94]等提出再在多级安全环境中的扩展。 而 Foley[95]利用 Unix 的简单保护机制及 Clark-wilson 模型的安全应用，在 Unix 系统中构造了中国墙安全策略，但该方法只适用安全要求比较低并且应用系统具有较高的结构化特性的情况。Jaeger 等[60]及 McCune[59]分别在虚拟机及分布式虚拟机环境下，使用中国墙的简单安全性质来约束隐形流。

在模型的实施方面近期的研究集中于利用中国墙模型的冲突关系来划分访问区域。例如 Katsuno[96]在分布式环境中实施细粒度强制访问策略时，使用中国墙标签将资源静态划分为不同的域，域内可以允许信息流动，其它资源可以选择加入某一个域，一旦选择不能在改动。Jaeger 等[60]讨论控制 VM 系统中隐形流时，提出采用 Caernarvon 策略，静态将资源划分为不同的隔离域，域间是信息隔离的，并探讨了采用中国墙模型动态产生冲突集划分通信区域的可能。Lin 等[97]使用中国墙模型提出粒度计算概念，用来静态划分客体约束信息流。在另外一些应用场景中，例如动态合作组织中实施访问控制时，也通常需要动态划分访问区域，允许信息在某一个域中流动。

2.4.3 BN 中国墙策略对隐形流的控制

在可信虚拟域间实施强制访问控制的目的是为了在虚拟机粒度上实现更强的隔离。 在 IBM 可信虚拟域的联合可信基上实施的强制访问控制策略是以 sHype[56]和 Shamon[59]系统中支持的访问控制策略为基础的。因此，下文将以这两个系统为例说明在可信虚拟域间进行隐形流控制对访问控制模型的要求。sHype 和 Shamon 使用类型控制模型（Type Enforcement Model）来描述被系统授权的虚拟机间通信。sHype 和 Shamon 并不有意包含隐通道，但却不保证没有隐形流。类型控制模型（简称 TE 模型）适合描述基于格的隔离安全策略，却无法控制通过隐通道传递的信息流。被禁止通信的两个虚拟机可能通过隐通道违反安全策略而造成信息泄露。sHype 和 Shamon 使用中国墙策略来控制这种风险：管理员给虚拟机打上标签，并在标签中定义不同的冲突集。在初始阶段，系统可以运行任意标签的虚拟机，而随后选择加载的虚拟机是有限制的，即该虚拟机的标签不能和已经运行的虚拟机标签相冲突。

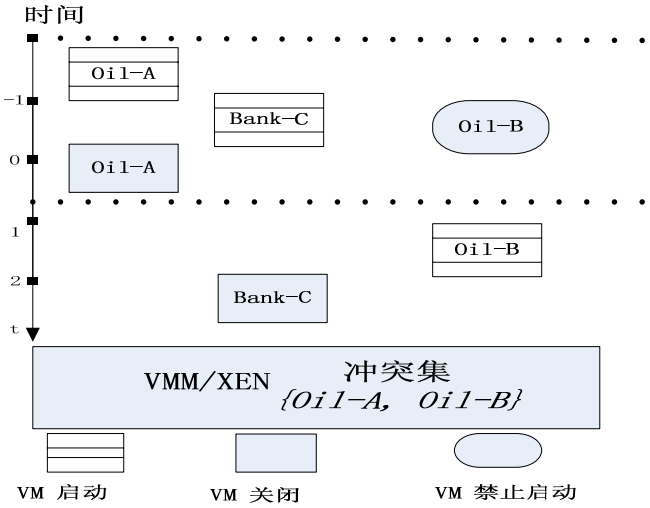

图 2.2 在 sHype 系统中使用中国墙策略控制隐形流示例

图 2.2 描述了在 sHype 系统中，如何使用中国墙策略控制虚拟机的启动来消除虚拟机间的隐形流[56]。sHyper 只使用中国墙策略的 BN 简单安全规则，如图所示：Oil-A、Oil-B 和 Bank-C 是三个虚拟机标签，其中冲突集为{ Oil-A， Oil-B }，这意味者标签为 Oil-A 的虚拟机（简称 Oil-A）和标签为 Oil-B 的虚拟机（简称 Oil-B）之间禁止有

任何的信息流，sHyper 将禁止它们同时运行来避免潜在的隐形流风险。当 Oil-A 停止运行时， Oil-B 才允许被载入。

sHype 系统假设只有两个虚拟机同时运行时才可能存在隐形流，如图 2.2 所示，当 Oil-B 被载入时，Oil-A 已经不再运行。因此，即使有隐通道的存在，也不能被 Oil-A 的恶意程序利用向 Oil-B 中的一个等待进程传递信息。然而 Oil-A 和 Oil-B 可以借助其他虚拟机间接的传递隐形流，考察图 2.2 中的例子，可以看出：在初始阶段，标签为 Bank-C 的虚拟机（简称 Bank-C）和 Oil-A 同时运行于同一个虚拟机监控器之上；在 Oil-A 关闭后，Bank-C 同 Oil-B 同时运行，间接的隐形流就可能通过 Bank-C 在 Oil-A 和 Oil-B 之间传递。

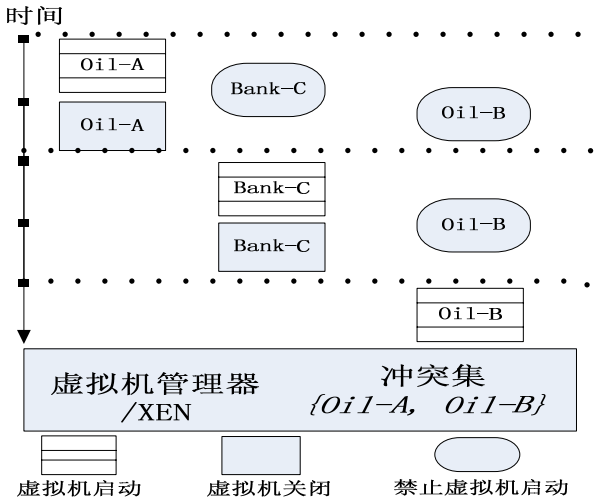

图 2.3 BN 模型对 VM 间接隐形流的控制

BN*特性禁止在不同的冲突类中进行写操作，因此可以在上述系统中使用 BN 特性来防止间接隐形流。图 2.3 描述了在 sHyper 系统使用 BN *特性防止间接隐形流的实施效果。初始阶段系统允许任意标签的 VM 运行，标签为{Oil-A，Oil-B}的 VM 在一个冲突集中，应该禁止它们之间有任何的信息流。因此在标签为 Oil-A 的 VM 运行阶段，禁止标签为 Oil-B 的 VM 启动。标签为 Bank-C 的 VM 同标签为 Oil-A 和 Oil-B 的 VM 处于不同的冲突类，但是依照 BN 模型的写规则，一旦主体对不同冲突类客体进行了读操作就禁止任何写操作。为了防止 Oil-A 和 Oil-B 之间通过 Bank-C 进行间接的隐形流传递(同时运行可能通过隐通道进行读写操作)，BN 模型禁止它们

同时运行。从图 2.3 中可以看出，使用 BN*特性可以消除虚拟机间的隐形流，但是需要事先安排虚拟机的启动顺序，这意味着在虚拟机运行以前，要根据标签的分配将资源绑定虚拟机上。

2.4.4 其他模型对隐形流的控制

基于格的访问控制模型也可以用于控制隐形流，但都需要加强约束条件。以 Caernarvon[61] 为例，该策略规定一个主体可以在一定的标签范围内读写，在这个范围内外可以上读下写。例如定义如下标签（机密，秘密，保密，公开），并规定它们的安全性依此递减。规定范围{秘密-保密}内允许信息流双向流动，而在这个范围之外的要满足上写下读的规则。即标签为秘密和保密的主客体之间可以相互读写，从机密到秘密或是保密只能读，而公开到秘密或是机密只能写。

不考虑隐形流，以在图 2.2 中定义的标签为例： 如果要规定一下 Bank-C 可以读 Oil-A 而写 Oil-B，可以将它们的标签如下排列（Oil-A，Bank-C，Oil-B）。如果考虑隐形流则单向的流约束被打破，因此 Caernarvon 策略的单向流约束的规则是无法用于隐形流的控制，但是 Caernarvon 双向流策略可以用于控制隐形流。不同于中国墙策略可以依据主客体的不同访问顺序来对信息流进行约束，使用 Caernarvon 策略控制隐形流，需要事先进行依据标签进行划分。一旦标签确定，访问的范围也就确定了，例如使用 Caernarvon 控制图 2.2 示例中的隐形流问题，可以事先将 VM 依照标签分组为{Bank-C+ Oil-B} {Oil-A} 或是 {Bank-C+Oil-A} {Oil-B}，只有虚拟机拥有的标签属于同一组内才可以同时运行。

2.4.5 隐形流控制需求

通过强制访问控制模型来约束虚拟机间的隐形流并不需要通过重新编码虚拟机监控软件来消除隐通道，而是通过这些策略来调度资源避免安全策略中禁止信息流的虚拟机间的资源共享，并提供给用户一个可以配置的方式来确定应该禁止哪些虚拟机间的信息流。

现有的访问控制模型可以非常有效地控制公开流（Overt Flows），然而在用于控制隐形流时，是以增强约束条件增加开销为代价。使用现有的强制访问策略来处

理隐形流，会使系统有太多的限制而缺乏灵活性和难以实施。例如采用 BN 中国墙模型的写操作约束或是基于格的 Caernarvon 模型来控制隐形流，通常需要静态地划分客体[60][96]。然而，在一些分布式环境下，事先对客体进行划分是不可能的，如在动态合作组织中，客体是动态加入的。在另一些应用中，事先不知道主体完成某项事务需要访问的客体，必须依据访问的需求进行动态地划分[60]。

因此，使用访问控制的方式控制虚拟机系统中的隐形流需要访问控制模型能为用户提供一个可配置的调度方式来管理虚拟机资源，并且能同时保持类似中国墙策略在控制公开流时的依据访问的需要进行选择的特性和类似格策略的能够将客体分成不同的访问集的特性。针对于这种需求，本章将通过对中国墙模型及其变种的信息流约束规则进行分析，提出一种新的中国墙策略。该策略能动态扩展冲突关系，在不同的冲突关系中依照动态联盟形成访问区域，使不同的区域不包含利益相冲突的数据集，信息可以在某一个区域中流动。同时，该策略具有与 BN 中国墙模型相同的安全目标，敏感数据不因主体的访问泄漏到利益相冲突的数据集中。

2.5 可信虚拟域间隔离的优先中国墙模型

2.5.1 利益冲突关系描述

BN 模型根据利益冲突关系划分利益冲突类，其中蕴涵的假设是利益冲突关系是数学上的一个满足对称、自反和传递的二元关系。用 CIR 表示利益冲突关系，O 表示客体集合。$CIR=\{(u,v)\}\subseteq O\times O$，一般情况下，CIR 是一个满足对称和非自反和非传递的二元关系[87]。例如描述二战时的国家关系，O＝{美国，德国，英国}，CIR=“敌对”，则美国和德国是敌对关系，英国和德国也是敌对关系。如果利益冲突关系满足传递性，那么美国和英国也是敌对的，这显然是不合理的。BN 中国墙模型根据利益冲突关系划分的利益冲突类通常情况下不是一个等价划分，存在一个企业数据集属于几个利益冲突类的情况，所以 BN 中国墙模型中的数据结构不能通过其定义的 CIR 得到。故保持 BN 中国墙模型数据结构的正确性，实质上是使用 CIR 的传递自反闭包代替 CIR，在数学上保证了冲突类划分的正确性，但却使模型的约束条

件太强，使得一些主体无法访问那些本身并不冲突的企业数据集，因此在本文中将保持 CIR 不变，采用新的数据模型。

2.5.2 数据模型定义

S 表示主体的集合，O 为客体集合(此处的客体对应 BN 中国墙模型数据模型中的企业数据集，因为主体对同一个数据集中的所有数据具有相同的访问权限，故此处不区分数据集内的单个客体)。

CIC 表示 O 的一个覆盖，对 $\forall K \in CIC$，K 中的客体彼此利益冲突即 $\forall o,o' \in K,(o,o') \in CIR$，本文把这样的一个覆盖称为利益冲突群[98]，以区别于利益冲突类。将与客体 o 相冲突的集合记为

$$CIS(o)=\{o' \mid (o,o') \in CIR\}$$

访问控制矩阵 $A(s,o)$（其中 $s \in S,o \in O$），$A \to \{-1,0,1\}$，如果矩阵元素取值为 1，表示 s 曾经访问过或者是有权访问该客体。如果矩阵元素取值为 0 表示访问权限还没有决定。矩阵元素取－1，表示不允许访问。用 $T=\{0,1,2\cdots,t,\cdots\}$ 表示时间指标。本文用 $R(t)$ 表示 t 时刻所有访问请求，$R_t(s,o,r)$ 和 $R_t(s,o,w)$ 分别表示 t 时刻的读写访问请求。$R_t(s,o,)=1$ 表示访问被访问规则准许，$R_t(s,o,)=-1$ 表示访问被访问规则拒绝。$A_t(s,o)$ 表示 t 时刻的访问矩阵，在系统的生命周期内将随着 $R(t)$ 而变化。

2.5.3 信息流约束

中国墙模型同等考虑了保密性和安全性，任何保密性和安全性的模型都要体现为信息流的约束。信息流可以通过主体对客体的访问直接或者间接地在客体之间或客体与主体之间传递。

如果系统在某个时刻 $t \in T$ 允许主体 s 在对客体 o 进行读操作，信息将从 o 流动到 s。如果在另一个时刻 $t \le t' \in T$，允许 s 对 o' 进行写操作，那么信息将经过 s，从 o 流动到 o'。用 DIF 表示这种直接信息流的传递。

定义 2.12 对所有 $o,o' \in O;t \le t' \in T$：

1) $oDIFs \Leftrightarrow \exists s \in S : R_t(s,o,r)=1$

2) $oDIFo' \Leftrightarrow \exists s \in S : R_t(s,o,r) \wedge R_t(s,o',w)$

BN 中国墙模型简单安全性质防止具有冲突关系客体之间的直接信息流，它的BN*特性对写访问进行如下式的约束。

$A_t(s,o)=1 \wedge \forall o' \in O \setminus \{o,o_0\} : A_t(s,o') \neq 1$

第一个约束条件允许写操作如果主体 s 访问过该对象，第二个约束条件暗示写操作只能发生在初始阶段，一旦一个主体曾经访问过另一个客体 $o \neq o_0$，它将不能获取任何的写操作权限。本文注意到 BN 模型的 BN*特性可以防止以下间接信息流导致的利益冲突，假设 o 和 o' 利益冲突，o'' 与两者都不冲突，简单安全性质防止 $oDIFs \wedge o'DIFs$。但是如果允许 $oDIFo''$，便会出现 $o''DIFs \wedge o'DIFs$，因为 o'' 中存在 o 的信息，这会违背中国墙安全策略。这种间接传递的信息流，使用 CIF 表示：

定义 2.13 对所有 $s \in S; o,o' \in O; t \leq t' \in T$

1) $oCIFs \Leftrightarrow$

$\exists s_1, s_2 \ldots s_k = s \in S; o_1, o_2 \ldots, o_{k-1} = o \in O;$

$\exists\ t \leq t_1 \leq t_2 \leq \ldots t_{2k-1} \leq t' \in T;$
$R_1(s_1, o, r) = 1 \wedge \forall\ i = 1, \ldots k-1:$
$R_i(s_i, o_i, w) = 1 \wedge R_{i+1}(s_{i+1}, o_i, r) = 1;$

2) $oCIFo' \Leftrightarrow$

$\exists s_1, s_2 \ldots s_k \in S;\ o = o_1, o_2 \ldots, o_{k+1} = o' \in O;\ \exists t \leq t_1 \leq t_2 \leq \ldots t_{2k} \leq t' \in T$

$\forall i = 1, \ldots k : R_{2i-1}(s_i, o_i, r) \wedge R_{2i}(s_i, o_{i+1}, w)$

Lin[87]在研究冲突关系的数学定义时对写规则做了更加严格的限制：

$A_t(s,o) \neq -1 \wedge \forall o' \in O, o' \neq o, o_0 : A_t(s,o') \neq -1$

$A_t(s,o') \neq -1$ 意味 $\exists o'' \in CIS(o) \wedge A_t(s,o'') = 1$

故通常情况下，写权限几乎是不被允许的。因此在 BN 和 Lin 模型中不存在 $oDIFo' \wedge o \neq o'$ 这种信息流，信息不能在不同的客体间流动。

定义 2.14 系统被称为冲突安全的，如果 $\forall s \in S; o,o' \in O, t,t' \in T:$ $oCIF_t s \wedge o'CIF_{t'} s \Rightarrow o \notin CIS(o')$

Volker[91]通过动态的扩充冲突关系，修改了写约束，保证系统是冲突安全的，规定如果主体 s 可以在时刻 t+1 写客体 o，应满足以下三个条件：

$$A_t(s,o) \neq -1$$

$$\wedge \forall s' \neq s : R_t(s',o,r) \in R(t)$$

$$\wedge \forall s' \neq s : A_t(s',o)=1 \wedge \forall o' \in CIS_{t+1}(o) : A_t(s',o') \neq 1$$

第一个条件说明 o 不与 s 曾经访问过的客体具有冲突关系，第二个条件说明 t 时刻没有其他客体正在访问客体 o，第三个条件说明访问过 o 的其他主体不能曾经访问过与主体 s 曾经访问过的客体相冲突。同时为了保持冲突安全，用 CIR_t 表示 t 时刻冲突关系，Volker 动态扩充冲突关系：

$$CIS_{t+1}(o) = \bigcup_{\{o' \neq o | A_t(s,o')=1)\}} CIS_t(o')$$

满足以上条件则写操作被允许，并且主体 s 可以从它访问过的客体 o' 中拷贝信息到 o 中，即信息可以从 o' 流动到 o 中。本文注意到一旦一个主体 s 曾经读过一个客体 o，并且写入另一个客体 o'，则在 o 与 o' 之间便存在信息流 $oDIFo'$，因此另一个主体 s' 不允许同时读取一个客体 $o'' \in CIS(o)$ 和 o'，因此主体 s' 的选择将因为 s 的写操作受到限制。一旦 $oDIFo'$ 或者 $oCIFo'$，则 o' 中包含有 o 的信息，要保证 o 数据不通过 o' 泄漏到利益冲突的客体中，就要动态扩展 o' 的冲突集，使 $CIS(o) \subseteq CIS(o')$。因此 Volker 模型实质上随着不同的访问序列，主体之间的访问权限互相制约。这和 BN 及 Lin 模型在本质上不同。

从上述分析可以看到，当实施中国墙模型和上述的变种时，要对写操作进行严格的限制。这使得一但系统中主体需要对不同的冲突类中的客体进行写操作，就要静态的划分访问区域，从而防止因为间接的信息流导致的利益冲突。

2.5.4 联盟关系

Volker 模型中，因为其他主体的写操作，使主体的访问受到限制，其通过动态的扩展冲突集，本质上通过写操作划分了访问范围。然而该模型区分读写操作，即对读写操作给予的访问权限不同，使这种划分只能在读操作以前，基于动态联盟关

系的优先中国墙模型的目的根据主体的访问在不同的冲突关系中动态地构建访问区域，因此需要给予读写操作同等的约束。

规则 2.1 对于所有的 $s \in S; o,o' \in O; t \le t' \in T$ 如果 $R_{t'}(s,o,r)=1$，则有 $t \le t' \in T$ $R_{t'}(s,o,r)=1$；如果 $R(s,o,w)=1$，则有 $t \le t' \in T$ $R_{t'}(s,o,r)=1$。

本文的规则允许读操作就允许写操作，反之依然，即允许 $oDIFo'$ 存在则 $o'DIFo$ 存在。

定义 2.15 如果客体 o 和 o' 通过主体访问存在信息流 $oCIFo'$ 或 $o'DIFo$，则称 o 和 o' 具有联盟关系，用 AIR 表示。

在 t 时刻，根据规则 2.1，如果 o 和 o' 具有联盟关系。则 $oCIFo'$ 与 $o'DIFo$ 同时存在，信息可以在 o 与 o' 互相流动，AIR_t 是一个等价关系。

定理 2.5 对所有的 $t \in T$：AIR_t 是 $O \times O$ 上的等价关系。

证明：由定义 2.14 和规则 2.1 可知：

$o_i CIF_t o_i \Rightarrow o_i AIF_t o_i$

AIF_t 是自反的

$o_i AIF_t o_j' \Rightarrow o_i CIF_t o_j' \Rightarrow o_j DIF_t o_i \Rightarrow o_j AIF_t o_i$

故 AIF_t 是对称的

$o_i AIF_t o_j \wedge o_j AIF_t o_k \Rightarrow o_i CIF_t o_j \wedge o_j CIF_t o_k \Rightarrow o_i AIF_t o_k$

故 AIF_t 是传递的。

此证。

推论 AIR_t 是 $O \times O$ 上的等价关系可以导出 O 上的一个等价分化，o 是 O 的一个客体（企业数据集），则 O 中与 o 具有 AIR_t 关系的全体客体称为 o 的联盟。用 $AIF_t(o)$ 表示，即：

$$AIF_t(o) = \{ o' \mid (o,o') \in AIF_t \}$$

定义 2.16 布尔矩阵 $C(c,q)$（其中 $c \in O, q \in O$），如果元素取值为真，表示客体 o_c 和 o_q 曾经被同一个主体访问过，$C_t = A_t^* \times A_t$。

注：C_t表示到 t 时刻，因为主体的访问，在客体间形成的直接信息流，并随 $t+1$ 时刻的请求$R_{t+1}(s,o)$而变化到C_{t+1}。如果用C_t^*表示C_t的传递闭包，则C_t^*表示到 t 时刻，因为主体访问，客体间形成的间接和直接信息流。C_t^*是 t 时刻AIF_t的关系矩阵。

在规则 2.1 的前提下，客体通过主体的访问，具有联盟关系，联盟是一个等价划分，随着主体的访问，客体被划分到不同的区域中，但必须保证主体的访问是冲突安全的，下面给出保证冲突安全的规则。

2.5.5 新模型的安全规则

系统的状态可由一有序三元组(A_t,R_t,AIF_t)来描述，系统状态随 $t+1$ 时刻的请求$R_{t+1}(s,o)$而变到$(A_{t+1},R_{t+1},AIF_{t+1})$，请求由系统的访问规则决定。

AIR_t根据主体的访问在客体中建立起来的等价关系，可以导出 O 上的一个等价分化实现动态地构建访问区域，要实现冲突安全，给出以下访问规则：

规则 2.2 系统在初始状态下，任一主体可以访问任何的客体

$$A_0(s,o)=\begin{cases}1: s=s_0 \vee o=o_0\\ 0: s\neq s_0 \wedge o\neq o_0\end{cases}$$

$R_0=\phi$， $AIF_0=\phi$， $C_0=CIR$

在初始状态下，没有任何主体访问过客体，此时$AIF_0=\phi$，允许任何主体进行访问。但假设在 t 时刻主体 s 对o'发起访问，根据规则 2.1，可以知道允许主体访问o'，就意味$o'DIFs$和$sDIFo'$，信息流是传递的。要实现冲突安全，仅仅检查o'与 s 访问过的客体 o 之间的冲突关系是不够的，还需要检查与 o 具有联盟关系的所有客体。同理o'中也可能含有与之联盟的其他客体内的数据，所以还需检查所有与o'联盟的客体。

动态扩展冲突关系意味着冲突关系在系统的生命周期内随着主体的访问请求而改变，如果 $t+1$ 时刻的访问被准许，必须同时扩展客体o与o'的冲突集：

$$CIS_{t+1}(o)=CIS_{t+1}(o')=\bigcup_{\{o'\neq o|A_t(s,o')=1)\}}CIS_t(o')\cup\bigcup_{\{o'\neq o|A_t(s,o)=1)\}}CIS_t(o)$$

这意味着具有联盟关系的客体具有相同的冲突集。所以一旦一个主体访问过某一个联盟的客体o，对另一个联盟的客体o'发起访问请求，就要检查属于两个联盟的所有客体之间是否具有冲突关系。即访问客体o与o'的主体的访问权限将因为先前主体访问操作在o与o'建立的联盟关系而受到限制。

规则 2.3 *t+1* 时刻，访问$R(s,o')$满足以下 2 个条件之一：

$\exists o \in O \wedge A_t(s,o)=1 \wedge AIF_t(o)=AIF_t(o')$ 或者 $\forall o \in O, A_t(s,o)=1$

$\forall o'' \in AIF_t(o) \wedge o'' \notin \bigcup\{CIS(o) | o \in AIF_t(o')\}$访问被允许。

如果主体 s 将要访问客体o'和它曾经访问过的客体 o 具有联盟关系，具有联盟关系的客体之间允许信息流动，因此允许访问。如果主体 s 将要访问的客体o'与它访问过的客体不在同一个联盟中，要将o'所属的联盟中每个客体同 s 访问过的客体所属的联盟中的每个客体进行冲突关系的检查。

例如，假设 S 表示主体的集合，O 表示客体的集合，其中：

$S=\{s_1,s_2,s_3\}$; $O=\{A,B,C,D,E\}$

表 2.1 利益冲突关系

客体对象	客体对象
A	C
A	E
C	A
C	D
D	C
E	A

客体间的利益冲突关系如表 2.1 所示：CIC 表示所有客体的一个覆盖，对$\forall K \in CIC$，K 中的客体彼此利益冲突

CIC={K1, K2, K3, k4}

K1= {A,C} K2= {C,D}

K3= {A,E} K4= {B}

与每个客体相冲突的集合记为：

CIS (A) = {C, E}

CIS (B) = { }

CIS (C) = {A, D}

CIS (D) = {C}

CIS (E) = {A}

假设有三个主体s_1,s_2,s_3，t 时刻的访问矩阵如下：

$$A_t=\begin{pmatrix}1 & 1 & -1 & 0 & -1\\ 1 & 0 & -1 & 1 & -1\\ -1 & 0 & 1 & -1 & 1\end{pmatrix}$$

因为主体的访问，t 时刻形成了两个联盟 AIF {A,B,D}，{C,E}。 假设s_1在 $t+1$ 时刻发起访问$R_{t+1}(s_1,D)$，依据规则 2.2，s_1曾经访问过 A，而$B\in AIF(A,B,D)$；因此该访问被允许。

相应访问矩阵为：

$$A_{t+1}=\begin{pmatrix}1 & 1 & -1 & 1 & -1\\ 1 & 0 & -1 & 1 & -1\\ -1 & 0 & 1 & -1 & 1\end{pmatrix}$$

若此时访问请求$R_{t+1}(s_3,B)$依据规则 2.2 只有$B\notin CIS(C)\cup CIS(E)\wedge A\notin CIS(C)\cup CIS(E)\wedge D\notin CIS(C)\cup CIS(E)$时访问才被允许，因为$A\in CIS(C)\cup CIS(E)$，同时$D\in CIS(C)\cup CIS(E)$，故此访问请求不被允许。客体随着主体的访问，被动态划分为两个访问区域{A,B,D}和{C,E}。

2.6 安全目标证明及其应用分析

2.6.1 新模型的安全目标证明

新变种在不同的冲突关系中动态地构建访问区域，信息可以在某一个区域中流动，但要保证与 BN 模型相似的安全目标：敏感数据不因主体的访问泄漏到利益相冲突的客体中（数据集）。下面将给出理论上的证明。

定理 2.6 对于所有的$s\in S;o,o'\in O;t\in T$，如果$\forall A_t(S_u,o)=0$，则$R_t(s_u,o)=1$。

证明：因为 $\forall A_t(S_u,o)=0$，说明该主体从未访问过任何客体，依据规则 5.3，$R_t(s_u,o)$不会与任何主体访问过的客体冲突，故 $R_t(s_u,o)=1$。

定理 2.6 说明对于主体 s，它最初的访问选择是没有约束的。

定理 2.7 对所有的$t\in T$:

$AIF_0=\phi$；

$AIF_0\subseteq AIF_1\subseteq\cdots AIF_t\subseteq\cdots$；

证明：由定义 2.12 和规则 2.1 可知，在初始状态下，没有主体访问客体，此时不存在信息流动，故 $AIF_0=\phi$。由 AIF 的定义可知，有 AIF 除初始状态外，访问请求只在规则 2.2 允许的情况下改变 AIF，故可证。

随着主体的访问，一旦主体选择了一个访问客体，就决定了它的访问只能沿着该客体所属的联盟，此联盟是由其他的主体访问形成的约束，或经过冲突检查后扩展该联盟。联盟动态扩展形成不同的通信区域，从而实现了对客体的划分。

定理 2.8 系统是冲突安全，对所有$s\in S;o,o'\in O;t\in T:\ oCIFo'\Rightarrow(o,o')\notin CIR$

证明: 由定理 2.7 可知$(o,o')\in AIR_{t+1}\Leftrightarrow(o,o')\in AIF_t$

$\vee\exists s\in S:R_{t+1}(s,o)\wedge\exists o''\in AIF_t(o'),A_t(s,o'')=1\vee\exists s\in S:R_{t+1}(s,o')\wedge\exists o''\in AIF_t(o),A_t(s,o'')=1$

因此如果$(o,o')\notin AIF_t$则必然在 $t+1$ 时刻

$\exists s\in S:R_{t+1}(s,o)\wedge\exists o''\in AIF_t(o'),A_t(s,o'')=1\ \ \vee\exists s\in S:R_{t+1}(s,o')\wedge\exists o''\in AIF_t(o),A_t(s,o'')=1$

由规则 2.3 可知 $t+1$ 时刻的$R_{t+1}(s,o)$

或者$R_{t+1}(s,o')$必然不会导致$(o,o')\notin CIR$，如果$(o,o')\in AIF_t$，因为 $AIF_0=\phi$，则必然存在时刻 k，满足$(o,o')\notin AIF_{k-1}$

$\exists s\in S:R_k(s,o)\wedge\exists o''\in AIF_{k-1}(o'),A_{k-1}(s,o'')=1\vee\exists s\in S:R_k(s,o')\wedge\exists o''\in AIF_{k-1}(o),A_{k-1}(s,o'')=1$

同理在 k 时刻的访问不会导致$(o,o')\notin CIR$因此系统是冲突安全的。

满足访问规则的联盟动态扩展形成不同的通信区域，敏感信息不会泄漏到具有冲突关系的客体中。

2.6.2 应用分析

如上文所述，使用现有的强制访问策略来处理可信虚拟域间隐形流，会使系统有太多的限制而缺乏灵活性和难以实施。例如使用 BN 中国墙策略和 Caernarvon 策略，只能依照前文所述的方式—静态地将虚拟机划分成不同的访问区域。静态划分意味着在系统运行前，就要预知 VM 的行为，从而将虚拟机划分成不同的访问区域。例如{Oil-A, Bank-C}， {Oil-B}; 或者是{Oil-A}，{Oil-B, Bank-C}。这种消除隐形流的访问控制策略的实质是消除资源共享，事先的划分意味着在虚拟机运行以前，要根据标签的分配将资源绑定到虚拟机上。这种消除隐形流的方式是以牺牲虚拟机动态的调度和聚合为代价的，不可避免地造成高昂的性能开销和低的资源利用率。

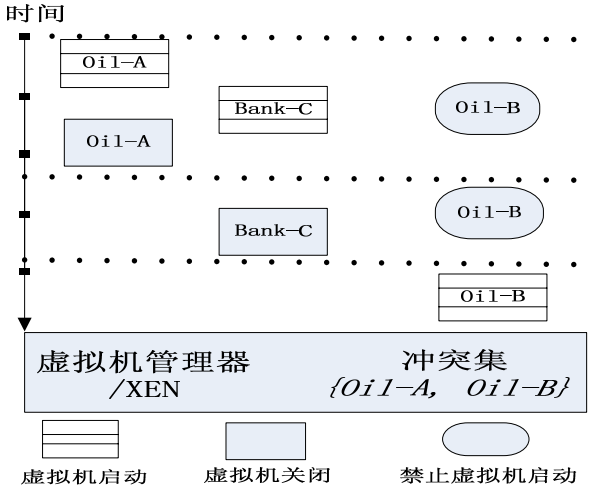

图 2.4 优先中国墙模型对 VM 间接隐形流的控制

Volker 模型中虽然对写操作的限制进行了修改，在不同冲突类（为了同 BN 中国墙比较，此处使用 BN 中国墙中对冲突关系的定义。PCW 中采用 Lin[87]对冲突关系更加严谨的数学定义，并不影响同其它模型的访问规则作比较）中，如果该冲突类的数据集没有被写操作过或被其它冲突类的主体访问过，则允许写操作，同时扩展该冲突关系。Volker 模型相比 BN 中国墙模型具有更大的灵活性，然而却无法用于控制虚拟机系统中的隐形流。因为隐形流的存在，只要不同冲突类的 VM 在一起运行过，就不能保证没有信息的流动。因此使用 Volker 模型控制隐形流类似于 BN 模型，同样需要对虚拟机进行静态地划分。

而本章提出的基于动态联盟关系的优先中国墙模型（PCW）同 Volker 模型相比，同等重要考虑了读与写操作，只要是跨冲突类的访问都进行利益冲突检查和动态扩展冲突关系。而后者只在写操作时，才进行冲突关系的扩展。采用本章的模型，在 sHyper 和 Shamon 系统中可以依据虚拟机运行时的访问需求建立客体间的联盟关系，从而动态扩展冲突关系。如图 2.4 所示，本章提出的模型不阻止 Bank-C 同 Oil-A 同时运行，只是在它们之间建立联盟，动态扩展它们的冲突关系，CIS（Oil-B）={Oil-A，Bank-C}，从而划分访问区域。

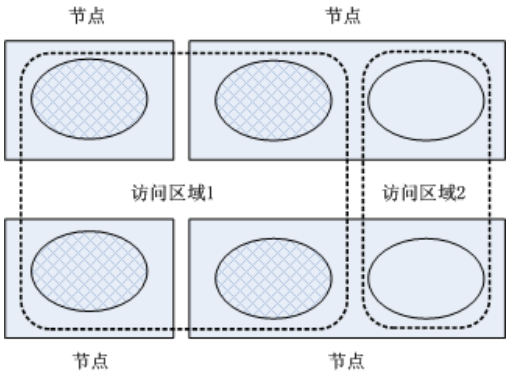

图 2.5 分布式联盟

PCW 模型虽然是为解决可信虚拟域中的隐形流问题而提出的，但作为一个访问控制模型，其应用不仅仅局限于虚拟机的场景中。可以代替传统的中国墙策略用于资源划分的场景中。例如应用于分布式联盟[99]--在分布式环境下将分布在不同节点的资源划分成不同的访问区域（如图 2.5 所示），并且在不同的区域中实施不同的安全策略，是在分布式环境下实施强制访问控制的研究热点[96][100]。通过分布式联盟，在不同的节点间提供安全信息的操作和交换，并使信息不会泄露到利益冲突的访问区域中。ALDC[96]直接利用中国墙利益冲突关系建立访问区域。PCW 对访问区域划分更为灵活，可以用于类似 ALDC 等系统中建立访问控制区域。

2.7 小结

本章的信任链模型和可信虚拟域间隔离模型分别给出了不同应用场景（客户端、服务端和分布式环境中）中构建虚拟机架构下可信计算环境的支撑理论。其中，信

任链模型通过实体间的依赖关系和可信集等概念给出了可信状态、可信度量及信任链的形式化定义和相关证明。该模型不依赖于具体的可信计算平台，具有普遍性。在本章中，使用该模型评估了现有的可信计算环境（基于静态可信度量根或是基于动态可信度量根）的信任链设计。

针对可信虚拟域间隐形流控制的需求，本章提出的基于动态联盟关系的优先中国墙模型（PCW），PCW 可以根据主体的访问需求建立联盟关系实现冲突关系动态扩展，从而将客体动态划分成不同的访问区域，允许敏感信息在不同的域内流动，但不会泄漏到具有冲突关系的客体中。本章给出了优先中国墙模型的形式化描述并证明了该模型具有和 BN 中国墙策略相同的安全目标。最后分析了该模型与 BN 和 Volker 模型相比在虚拟机隐形流控制上及分布式联盟中划分访问区域的优势。

3 基于虚拟机架构的透明信任链机制

本章讨论基于虚拟机架构的透明信任链机制—TCT，TCT 能够将信任链扩展到应用程序层，并保持对操作系统透明，同时 TCT 可以在用户期望的可信环境遭到破坏时保护其指定敏感数据。

3.1 研究背景

第二章中信任链模型的主要思想是将系统分解成一系列的实体，让一些实体去检验其它的实体。至于如何分解配置，如何检验实体，由哪一个或是哪一些实体进行完整性检查，以及为什么要信任这些实体等问题，则同可信平台的软硬件架构相关，在不同的平台上有不同的实现，但通常有两种处理方式：可信启动和安全启动。可信启动是可信平台记录启动的软件配置，安全启动则将启动的软件配置同一个正确的配置相比较，如果出错就终止，并保护正确配置下的隐私数据。

可信组织 TCG 定义了一套可信启动标准，描述了如何进行完整性度量和证明。TCG 方式的可信启动度量系统的软件栈并提供对外证明的能力，系统是否值得信任是由平台外的依赖方来判定。TCG 的可信度量从系统加电开始，首先将控制权交给 BIOS 中固化的一段代码（信任根），信任根通过计算哈希值的方式度量 BIOS 中其它部分，并把度量值存储到 TPM 的 PCR 中，然后将控制权移交给被度量的代码，这个过程一直进行下去直到操作系统启动完毕。

TCG 的可信启动只定义了度量加载的固定启动序列，而没有定义如何将信任链扩展到应用程序层。实际上当操作系统启动后，还有大量可执行的代码被加载（内核模块、二进制的共享库、脚本、插件、进程等），这些被加载的代码依据实际的需求而没有固定的序列。依据本文第二章的分析，对于依赖方 A，在时刻 t，实体 e 是否值得信任取决于 $Dep_t(e) \subseteq Trustset(A)$。因此，即使操作系统属于依赖方的可信集，还需要确定从操作系统启动完毕到依赖方验证时刻 t 加载的软件实体是否属于依赖方的可信集（因为操作系统的弱隔离性，通常需要假定 $Dep_t(e)$ 中包括所有的操作

系统实体和其他进程实体）。如果依照 TCG 定义的可信认证方式，提交的度量报告中应该包括从系统加电到依赖方验证时刻 *t* 所有被加载进程的哈希值。

在操作系统启动后，如何将信任链扩展到应用软件层，是 TCG 可信计算研究中的热点。国际上进行了一系列研究，比如，前文提到的 Dartmouth 大学 PKI 实验室的研究项目 BEAR[39][40]、 IBM 研究中心 Sailer 等人的 IMA 系统[41] 、Pennsylvania 大学 Jaeger 等人的 PRIMA 系统[65]等，这些都是极具代表性的工作，也体现了最新的研究水平。

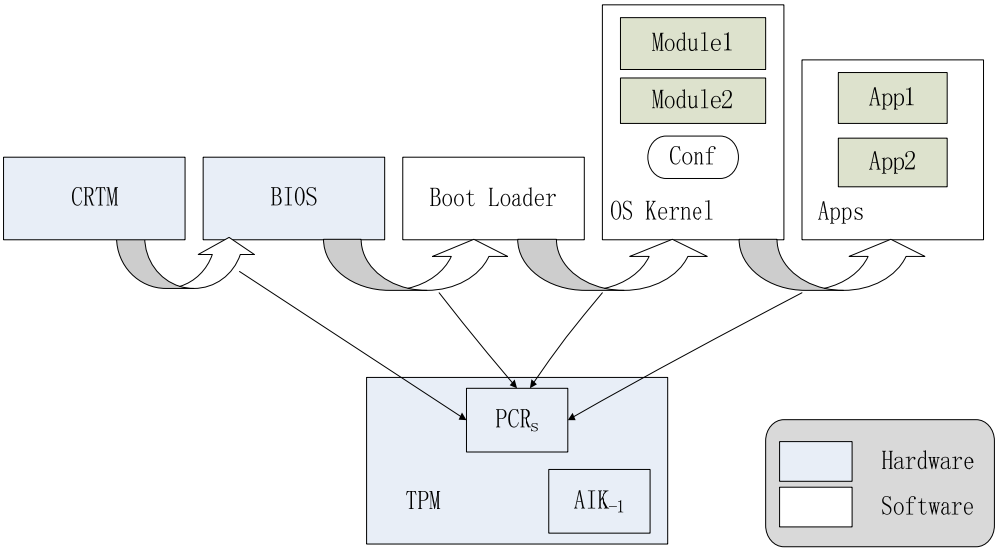

图 3.1 IMA 的完整性度量方式

然而这些系统都是通过修改操作系统内核的方式（对某些系统调用增加钩子函数）来度量操作系统启动后载入的可执行代码（如图 3.1 所示）。这种方式具有很多缺陷：首先修改操作系统的内核意味着这些方法对操作系统是不透明的，这使现有的操作系统无法支持应用程序级的完整性度量和认证。其次，该方式以操作系统为信任基（TCB），操作系统代码庞大，系统漏洞数量多且很难发现。第三，上述的方式只能为依赖方提供平台真实性的证明，而不能构建具有机密性保护能力的可信计算环境，难以满足分布式的应用。例如，远端的依赖方即使在 *t* 时刻验证了平台的完整性，也无法保证在 *t* 时刻后该平台的完整性受到破坏时，依赖方同该平台实体的通信或存放在该平台上敏感数据的安全性。同时虚拟化技术迅速商用化，成为例如

云计算等分布式应用场景中的基础架构，这也迫切需要在虚拟机监控软件层为其上的应用提供透明的安全支撑。

针对上述问题，在第二章模型基础上，本章给出一种基于虚拟机架构的对操作系统透明的信任链和机密性保护机制（TCT）。TCT 在虚拟机监控软件中度量操作系统以及在操作系统启动后加载的可执行代码，并允许依赖方通过哈希值的方式指定其信任的计算环境，当该计算环境遭到破坏后保护依赖方的敏感数据。

3.2 真实性和机密性需求

TCT 的设计目标是能够在虚拟机监控软件中提供虚拟域内运行程序和数据的真实性证明及保护其机密性。

3.2.1 真实性证明

对于依赖方 A， 实体 e 是否值得信任取决于在验证时刻 t， $Dep_t(e)$ 是否属于 $Trustset(A)$，不同的依赖方有着不同的 $Trustset(A)$。在一个具体可信计算环境中，某个时间 t，当依赖方 A 需要验证计算环境 $\&_{CE}$ 的系统状态 S_t 或是某一个实体 e 是否可信时，是由这个计算环境 $\&_{CE}$ 中的某一个实体来提供信息传达计算环境 $\&_{CE}$ 的系统状态 S_t 或是某一个实体 e 是否值得信任。

本文在定义 2.8 中给出了可信证据集的概念，在可信计算的工程实践中是通过提供给远端的依赖方（一些文献中称为挑战者）计算环境 $\&_{CE}$ 中运行代码的真实性证明，而让依赖方来判定平台是否值得信任。因此定义 2.8 的可信证据集可以具体化为以下概念：

定义 3.1. 对依赖方 A，$Trusted(A)$ 表示某些实体身份证明的集合，A 信任拥有这些身份证明的实体行为。

证书和完整性是用于度量实体真实性的主要现实标准，证书通常由可信的第三方实体颁发，比较成熟的解决方案有 PKI 方案和 DAA 方案。而可信计算组织（TCG）

使用度量实体完整性的方式。依赖方 A 通常了解在一个计算环境 $\&_{CE}$ 中，哪些实体是可以信任的，它们对证书或是度量其它实体的操作行为符合依赖方 A 的期望。

定义 3.2. $Trace(E,t)$ 表示由 $\&_{CE}$ 中某个实体提供的，在时刻 t 前载入的实体身份证明的集合。

依赖方 A 对 $\&_{CE}$ 的系统状态 S_t 或是某一个实体 e 是否信任的判定算法是由 $Trusted(A)$ 和 $\&_{CE}$ 中实体提供的实体身份证明集合 $Trace(E,t)$ 所决定的。

定义 3.3. 可信状态认证方案 Vailidate 是这样一个机制，依赖方 A 在时刻 t 依据可信集 $Trusted(A)$ 和计算环境 $\&_{CE}$ 提供的可信证据集，可以判定系统是否处于可信状态。

定义 3.4. 对于任意实体 e，具有合法信任集的依赖方 A，在任意时刻 t 有：$Validate(Trace(E,t), Trusted(A)) \Leftrightarrow Dep_t(e) \subseteq Trustset(A)$，则 $Validate$ 是可靠和完备的。

由定义 3.4 可知，一个验证机制必须能够提供足够的信息使依赖方可以得出计算环境 $\&_{CE}$ 的系统状态 S_t 或是某一个实体 e 是否可信。对于基于实体真实性的验证机制，一个隐含的假定是真实性可以无损地度量实体的行为。在这样的假定下依照第二章中给出的信任链理论可以构造出满足定义 3.4 的验证机制。

3.2.2 机密性需求

定义 3.5 $\&_{CE}^{pt}$ 表示从系统加电到时刻 t 前所有加载程序的哈希值序列。

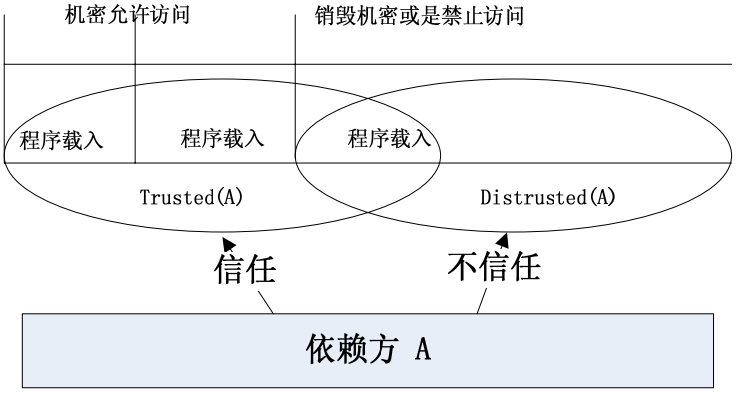

图 3.2 机密性需求

定义 3.6 依赖方 *A* 的机密性需求：

假设：依赖方 *A* 是计算环境 $\&_{CE}$ 中的合法用户，在时刻 t，$\&_{CE}^{pt} \in Trusted(A)$，*A* 在系统状态 S_t 下初始化其保护数据 *D*；$p_{t1}, p_{t2}, ..., p_{ti}, ...$ 表示在 t 时刻后加载到系统的程序，$\&_{CE}^{pt_1}, \&_{CE}^{pt_2}, ..., \&_{CE}^{pt_i} ....$ 表示相应的被加载程序的哈希值序列；

如果 $\&_{CE}^{pt_1}, \&_{CE}^{pt_2}, ..., \&_{CE}^{pt_i} ....$ 中，$\&_{CE}^{pt_k}$ 是第一个不属于 $Trusted(A)$ 的序列，而 $\forall t' < t_k, \&_{CE}^{pt'} \in Trusted(A)$，则在时刻 $t'' \geq t_k$ 时，数据 *D* 被销毁或是禁止访问；只有在时刻 $t \leq t'' < t_k$ 时，数据 *D* 才能被访问。

定义 3.6 给出了依赖方对可信计算环境机密性需求的形式化描述，即对于依赖方 *A*，在其信任的环境中创建的数据只能在其信任的环境中才能被访问，当载入的实体不被依赖方信任时，依赖方在其信任的环境中的秘密应该被保护起来禁止任何进程访问或是被销毁（如图 3.2 所示）。

3.2.3 可信基（TCB）需求

在虚拟机架构下，虚拟机监控器本身自然被认为是可信基（TCB），虚拟机监控器处于软件栈的最底层，监控其上软件栈的变化，因此需要将完整性度量和机密性保护的控制代码及判定策略集中存放虚拟层。然而这将引发一个新的问题：假设虚拟机监控器实体 *A* 被重新载入为 *B*，因为其下的 BIOS 代码没有虚拟机监控器可信更新的支持，因此 *A* 要自己执行载入过程，这将不可避免导致 $BDep(A)$。

A 可能有欺骗行为，例如不安装正确的代码。要解决这个问题必须使虚拟机监控器软件本身具有机密性 (“敌手无法看到虚拟机监控软件的秘密”) 和控制性 (“敌手无法改变虚拟机监控器运行”)的特性[31]，即使平台的所有者也不能破坏这两个特性。然而软件本身，不可能具有这种特性，最好的方式是能够使虚拟机监控器运行于类似 IBM4758[31]的安全协处理器上，这要求虚拟机监控器为上层应用提供可信计算环境的同时其首先要运行于可信环境之中。因为兼容性和昂贵的价格，运行于类似 IBM4758 的安全协处理器实现上几乎是不可能的。现有的商用可信平台（符合 TCG/TPM 规范）并不能提供 IBM4758 的安全协处理器的安全启动的功能，因此在

现有的商用可信平台上使用本章的机制，需要采用 TCG 可信启动的方式建立从硬件可信根到虚拟机监控器的信任链，而由虚拟机监控器将信任链扩展到其上的操作系统及应用程序层。

3.3 透明信任链的功能设计

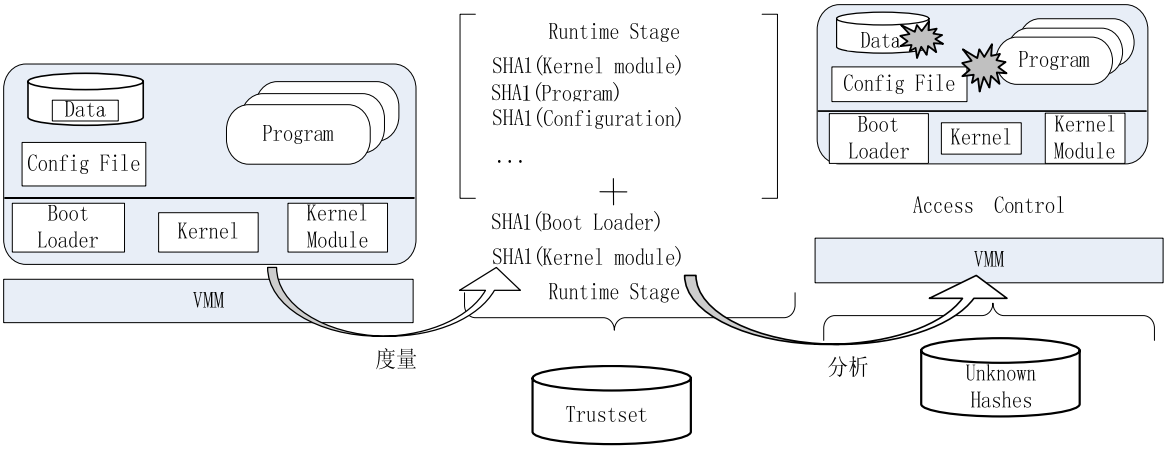

图 3.3 TCT 功能示例

要实现 3.2.1 和 3.2.2 的需求，以虚拟机监控器为可信基，将控制代码和策略判定集中于虚拟层，虚拟机监控器应具备以下功能（如图 3.3 所示）和特征：

度量--虚拟机监控器决定虚拟域中那些部分需要被度量及什么时候度量。

完整性证明和机密性保护--虚拟机监控器能够允许被授权的依赖方验证虚拟域内计算环境的完整性，并指定其信任集和保护数据。虚拟机监控器应该能够依据度量结果和合法依赖方的信任集，对其需要保护内容进行访问控制。

透明性--上述的功能不需要虚拟机内的客户操作系统和进程提供任何支持。

度量范围与时间 虚拟域中的用户进程无疑依赖于下层的操作系统，操作系统的所有模块都应该被度量。操作系统中的进程可能是相互依赖的，同时因为操作系统提供的进程间隔离性比较弱，除非是在具有强制访问控制的操作系统下，所有的进程都可能相互影响，因此所有代码不管它是由操作系统，动态加载器，或是程序代码加载的都需要被度量。在强制访问控制机制下，如果假定操作系统提供的进程间

隔离能够满足系统的安全目标，度量的范围可以缩小到进程的功能依赖集。对于一个程序的结构化数据（例如某个程序的视频文件），如果其内容具有可识别的完整性，应该如同程序可执行代码那样进行同样的度量。对于非结构化的动态数据（例如某个进程的缓冲数据），其内容没有一个完整的可识别语义，其完整性依赖于某些产生和修改它的进程的完整性。

操作系统的功能模块大多在其启动阶段被加载。在虚拟机架构下，虚拟机监控器并不直接载入操作系统，而是提供操作系统启动阶段位于 BIOS 中的启动代码，并将控制权移交给这段代码。因此引导代码和 OS 镜像都应该在虚拟机启动阶段被度量。在虚拟机运行阶段，大量可执行的代码将依据实际的需求被加载（内核模块、二进制的共享库、脚本、插件、程序等），这些代码应该在其被加载时度量。

完整性证明和机密性保护 度量采用 TCG 的 SHA1 哈希值方式，对于一个合法的依赖方来说，虚拟机监控器能够为其提供虚拟机内从引导时刻到验证时刻所有加载代码和结构数据的 SHA1 哈希值序列。如果依赖方需要验证虚拟机监控器自身的可信性，则虚拟机监控器还需要提供其自身的度量值（TCG/TPM 的可信报告根可以保证这些度量值的真实性）。

TCT 不仅仅提供一个类似 BEAR 和 IMA 的完整性证明功能，TCT 还为合法的依赖方提供机密性保护。这允许依赖方提供期望的 SHA1 哈希值序列来表示其信任集和指定其需要保护的数据。虚拟机监控器在虚拟机客户操作系统启动时和运行中度量相关的代码和数据，并同依赖方提供的信任集中的哈希值相匹配，当出现不匹配的情况，保护依赖方的被保护数据不被访问。依赖方的数据在其期望的完整性被破坏之前可能已经被读到了内存。虚拟机监控器需要同时控制对磁盘和内存的访问。

透明性 透明性意味着 TCT 的功能实现不需要对现有的操作系统和应用程序进行任何的修改。上述功能的实现需要在虚拟机监控器中来解析操作系统的操作语义，并能控制操作系统对磁盘和内存的操作。虚拟机监控器处于客户操作系统的下层，能够获取的信息都是低级语义信息。例如对于磁盘操作，虚拟机监控器获取的可能是读写的物理块号或者是对虚拟磁盘端口的访问指令和参数（取决于虚拟机监控器的实现方式）。而虚拟机监控器给依赖方的完整性报告或是从依赖方处得到的可信

集是以高级的语义表示的，例如目录文件等。虚拟机监控器需要能从这些低级的语义中恢复出高级语义，反之亦然。

3.4 透明信任链的主要机制

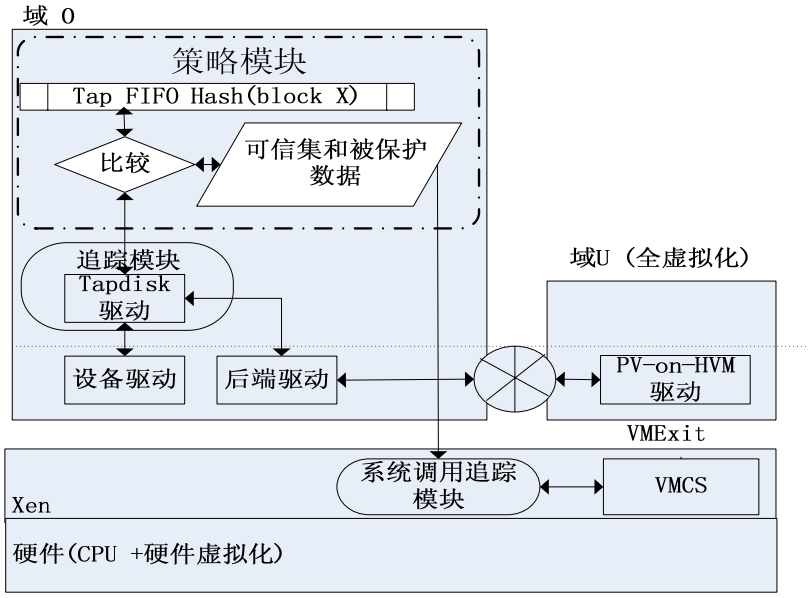

图 3.4 基于 Xen 的 TCT 实现

本节给出 TCT 在支持全虚拟化的 X86 平台上，基于 Xen 虚拟机监控器的原型实现。如图 3.4 所示，所有的度量操作和磁盘访问控制在 Xen 的管理域 dom0 中进行，而监控程序载入操作及保护内存敏感数据则是通过在虚拟机监控器中截获虚拟域内操作系统的系统调用来实现的。度量操作和磁盘访问控制利用了 Xen 的 Blktap 架构，监控程序载入操作和内存敏感数据保护利用 X86 快速系统调用机制和 Xen 内存管理子系统。

TCT 的原型包括一系列的功能模块：追踪模块（Trace Module--TAM)，系统调用追踪模块（System Call Tracer--SCT)，和策略模块（Decision Making Engine--DME)。TAM 收集所有磁盘操作信息以及完成度量和控制磁盘的操作，SCT 收集和过滤系统调用参数及实施内存访问控制，DME 依据 TAM 和 SCT 收集的信息做出度量或是访问控制的决策。

3.4.1 度量和磁盘访问控制

Xen 的 I/O 系统虚拟化可分为分离驱动模式（Split Driver Model），仿真模式（Emulational Device Model）和直接分配模式。其中分离驱动模式最为简单高效，但在 Xen 的全虚拟化域中需要安装特定的支持半虚拟化的驱动（PV-on-HVM driver）。TCT 原型系统中采用的是分离驱动的模式。在这种模式下驱动分为前后端，前端驱动位于虚拟域 domU 中，而后端驱动位于 Xen 的特权域 dom0 中。前后端的通信依赖于共享的异步 I/O 环、共享内存页和被称为 XenBus 的控制结构。对于磁盘设备，虚拟域 DomU 中对文件读写操作的系统调用请求被该域的操作系统内核转换为对磁盘块的读写请求。这些请求由位于 DomU 内核模式中的前端驱动传递给位于 Dom0 内核模式中的后端驱动，由后端驱动来访问真正的磁盘。

Xen 3.0.3 后引入了一种新的块设备架构-- Blktap 架构（如图 3.5 所示），Blktap 架构允许驱动实现为用户态的程序。这使 TCT 原型系统可以利用 Blktap 架构，在不改变 Dom0 内核的情况下，实现度量和磁盘访问控制。

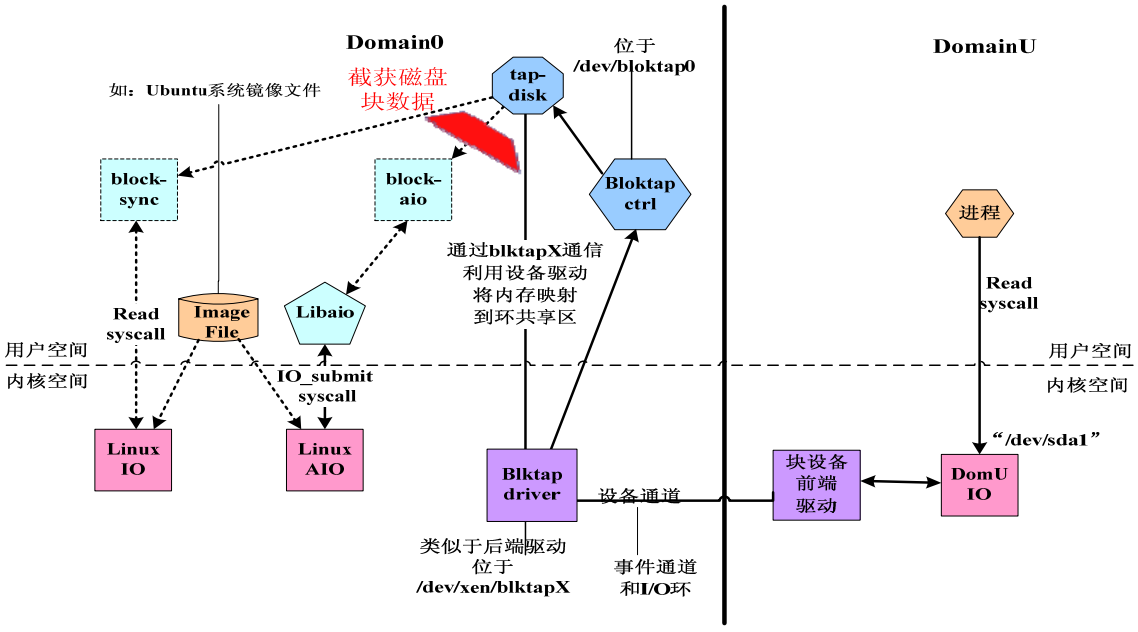

图 3.5 基于 Blktap 架构的 I/O 控制

TAM 在 Blktap 架构中增加钩子函数，在 Dom0 的用户态中拦截虚拟域 DomU 的磁盘操作。对于不同的磁盘操作，TAM 的处理方式不同。TAM 并不阻塞虚拟域的

读操作请求，而是将截获的操作请求信息发送给 DME 模块。这些信息中包含请求的扇区块号、读写的扇区长度等参数。TAM 等待 DME 的决策结果，如果在 DME 的决策结果做出以前，访问磁盘的读操作请求已经返回，这时需要阻塞 Tapdisk 将该缓冲区中的数据提交给虚拟域。TAM 需要依据 DME 的决策结果做出不同的响应： (1) 度量—TAM 允许 Tapdisk 返回虚拟域的读请求并将 Tapdisk 缓冲区中的数据拷贝到度量缓冲区中或是调用 Blktap 的异步 I/O 函数依据 DME 给出的参数读取特定的数据到度量缓冲区中。TAM 度量缓冲区中的内容，将哈希值返还给 DME；(2) 正常操作--TAM 允许 Tapdisk 返回虚拟域的读操作; (3) 拒绝– TAM 清空 Tapdisk 缓冲区，返还一个读操作错误给虚拟域。

TAM 阻塞任何的磁盘写操作，发送写操作参数给 DME（同读操作时发送给 DME 的参数相同），依据 DME 的反馈允许或是拒绝写操作。

3.4.2 系统调用追踪

在运行阶段，DME 需要根据操作系统的操作语义来决策何时进行度量和何时进行访问控制。SCT 通过截获操作系统的系统调用来解析操作语义。

Xen 虚拟机监控器对全虚拟化的支持依赖于处理器的硬件虚拟化扩展，Xen 虚拟机监控器运行于硬件虚拟化的根态（VMX-root model），而其全虚拟化域的操作系统运行于硬件虚拟化的非根态（VMX-non-root model）。全虚拟化域内的系统调用并不会引起运行态的转换而陷入 Xen 监控器中，因此，如果要在虚拟机监控器中追踪系统调用，并获取系统调用的参数，必须通过某些设置使系统调用可以引起状态的转换。在本章的原型中，SCT 通过对 X86 快速系统调用使用的一个处理器寄存器进行设置，使系统调用引起页错误从而陷入到 Xen 虚拟机监控器中。

X86 的快速系统调用机制使用 SYSENTER 指令使处理器从用户态跳转到内核态预先给定的地址继续执行。这个地址被存储到一个名为 SYSENTER_EIP_MSR 的特殊的寄存器中，这个寄存器只允许在处理器处于内核态时才能被设置。当用户态的程序需要系统服务时，它指定系统调用号和相关的参数然后调用 SYSENTER 指令。SCT 将 SYSENTER_EIP_MSR 值设置为一个不存在的地址，并将原有的地址保存在

Xen 的地址空间中。当系统调用发生时，这将引起页错误从而使处理器转入根态。SCT 比较页错误的地址，如果与在 SYSENTER_EIP_MSR 中设置的值相同，就意味者发生了系统调用。

SCT 没有必要追踪所有的系统调用，实际上只需要追踪与文件操作和程序及模块载入相关的系统调用即可，如 READ、WRITE、INT_MODULE、EXECVE 和 FORK 等。在运行阶段，可加载模块的动态载入通过 INSMOD 调用，而新的进程替换已有的代码通过 EXECVE 系统调用。

SCT 还需要获取上述系统调用的相关信息，例如可执行文件的路径或是系统调用参数等。这需要 SCT 理解客户操作系统的内核结构和操作语义。以 Linux 操作系统为例，如果要获取可执行文件的绝对路径等信息，需要获取当前进程的 task_struct。SCT 可以从 ESP 寄存器中获取 CPU 栈指针，该指针用来存放栈顶单元的地址。在 80x86 系统中，栈起始于末端，并朝这个内存区开始的方向增长。ESP 指向当前栈顶 0x0111a878 而 thread_info 结构是从 0x0111a000 开始存放。如果 thread_info 的长度为 4KB，那么将 ESP 的低 12 位屏蔽掉就可以得到 thread_info 的基址， thread_info 的第一个字段指向 task_struct 的指针，从而可以获取 task_struct 内存放的信息。对于系统调用的参数，可以直接访问相应的寄存器如 ebx, ecx, . . .等通用寄存器。这些调用的信息都发送给 DME 模块，由其来做出相关的决策。

3.4.3 内存访问控制

如果依赖方的被保护数据已经被读到内存，而此时其信任的计算环境完整性遭到了破坏，SCT 将依据 DME 的决策，对这些数据进行保护。最简单的保护方式就是重启该虚拟域，但是考虑一个虚拟域可能为多个依赖方服务，TCT 需要能够保护释放到内存的被保护数据。

TCT 原型系统通过 SCT 的系统调用追踪功能来获取这些被保护数据在内存中的位置信息。内存数据访问控制是通过 Xen 的影子页表机制实现的。 Xen 中的影子页表包含两类页表: 被客户机操作系统控制的客户页表 (GPTs)和被 Xen 控制的影子页表(SPTs)。影子页表完成实际的虚地址到机器物理地址的转换。为了实现对被保护数

据内存的访问控制，SCT 需要控制 GPTs 到 SPTs 的转换，同时创建一个被保护页表(PPTs)将被保护页的物理页表的入口指针存放于这个页表中（这个页表的其它位置置空）并清除影子页表中相应的入口指针然后刷新 TLBs。基于上述的设置，访问被保护的页表需要额外的 CR3 切换，类似于下一章 Cherub 架构中的双影子页表机制（详见 4.5.3.3）。

这种技术可以提供页面级别的保护，但是在很多情况下被保护的数据同其他数据存放在同一个物理页。SCT 通过修改 Xen 的页错误处理函数，提供字节粒度的数据保护。当对被保护数据所在页面访问造成页错误后，SCT 可以从处理器的 CR2 寄存器中获取访问数据的虚地址，并同 DME 模块中保存的被保护数据的虚地址进行匹配，如果匹配则禁止访问。

3.4.4 磁盘语义逆向解析

在虚拟机监控器中，大部分被 TAM 捕获的磁盘信息都是底层的操作信息，这些信息同具体的硬件架构和操作系统相关，而依赖方的可信集及被保护的数据都是由高级语义描述（在文件系统层次）。DME 面临的最大挑战是克服磁盘块操作和高层文件系统操作之间的语义鸿沟。DME 采用磁盘语义逆向解析的技术来解决这个问题。

磁盘语义逆向解析的核心是从底层的磁盘块操作中恢复出目录和文件操作语义，建立这些底层磁盘块到上层文件的反向映射。如图 3.6 所示，文件系统将高层文件划分成逻辑块，并维护逻辑块到物理块的映射。磁盘语义逆向解析正好相反，它将磁盘块映射到上层文件。显然，磁盘语义逆向解析和客户操作系统当前文件系统格式是紧密相关的。

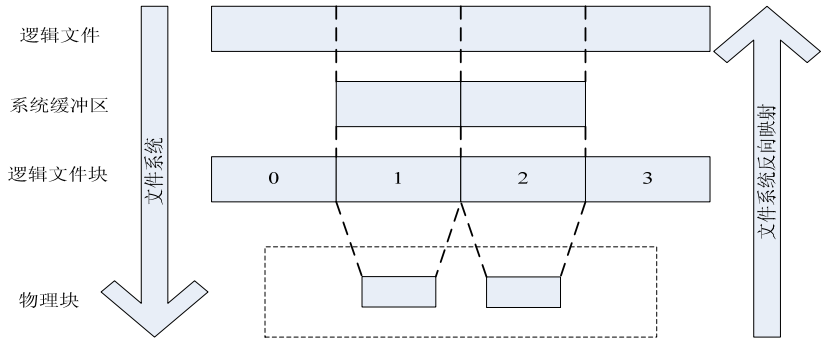

图 3.6　文件系统反向映射

以 ext2 文件系统为例，它由一个引导块和重复的块组构成，每个块组又由超级块、组描述符表、块位图、索引节点位图、索引节点表、数据区构成。DME 需要在它的内存空间中建立类似操作系统虚拟文件系统的数据结构，将可信集及被保护的数据使用目录结构描述的信息转化成对应的磁盘块号，并将要监控的磁盘块存放在对应数组中，从 TAM 传递来的磁盘块读写操作信息，可以在这些数组中进行匹配来推导出高级的操作语义。例如假设文件/etc/init.d/rc 对应磁盘块 211105、211106，文件/etc/profile 对应磁盘块 223236。如果 TAM 传递来信息对磁盘块 211105 进行了读操作，那么可以推断这次 I/O 的高层语义是读文件/etc/init.d/rc。

3.4.5 决策算法

DME 维护着依赖方的可信集和保护数据的目录信息，并依据 TAM 和 SCT 传递的信息做出度量或是访问控制的决策。为了上述目的，DME 创建了三个数据结构来维护这些信息。trust_inte_file 和 prote_file 分别记录着依赖方的信任集和被保护数据所在目录及数据自身占据的所有物理块和哈希值等信息，而 mem_pro_file 记录着被保护数据在内存的地址信息。

当目标虚拟机启动时，DME 首先将 TAM 发送来的启动模块代码和操作系统镜像的哈希值同 trust_inte_file 中的哈希值进行比较。如果不匹配，意味着启动的操作系统不能满足依赖方对系统完整性的需求。在 prote_file 中该依赖方的被保护数据的访问允许位将被置 1，即不允许访问。操作系统启动后，DME 从一个先进先出的队列中读取 TAM 发送来的磁盘操作物理块信息，并在 trust_inte_file 和 prote_file 中进行匹配。如果在 prote_file 中命中意味着磁盘操作的是被保护的数据，DME 检查访问该数据的访问允许位是否被置 1。如果被置位，则不允许访问。DME 发送拒绝访问的决策给 TAM。如果没有被置位，DME 还需要检查度量缓冲区是否为空，因为度量操作和磁盘操作是异步进行的，DME 必须等待直到所有的度量操作完成才能做出决策。如果被保护数据允许被访问，对于读操作，DME 在 prote_file 中设置一个标志位标明该数据已经被读入内存；对于写操作，DME 记录写操作完成后与被保护数据相关的改动物理块的信息。

如果 TAM 发送来的磁盘操作物理块号在 trust_inte_file 中命中，这意味着磁盘 I/O 访问的是可信集。为了减小性能的开销，对读操作，TAM 将缓冲可信集中的文件的度量结果，只有第一次访问或是被修改过，才会重新度量。如果该文件需要重新被度量，DME 发出度量指令给 TAM，指令中包括该文件占据的所有的物理块号。度量的结果将同 trust_inte_file 中保存的值进行匹配，如果命中，说明载入的文件满足该依赖方的完整性需求，反之，保护文件的访问允许位将被置 1。对于不属于可信集的读操作，TAM 同时依据 SCT 的信息判断是否是载入内核模块或是载入可执行的代码的操作，如果是，保护文件的访问允许位同样将被置 1。接着 DME 将判断是否保护文件打开标志位被设置，如果是，这标志被保护的数据已经被读入了内存，DME 将发送内存保护访问控制指令给 SCT。

3.5 应用和性能评测

基于虚拟机架构的透明信任链机制可以应用于类似于网格和云计算等多重独立软件授权环境下来保护上层用户的敏感数据和系统的完整性，本节将 3.3 节中实现的原型系统应用于 Nimbus[72] 中来说明 TCT 对增强云计算安全的有效性。TCT 的功能依赖于虚拟机监控器对上层操作系统的操作进行解析和控制，因此对操作系统 I/0 操作和系统调用会带来一定的开销。性能测试将着重对系统在这两个方面的时间开销进行测量并同未采用 TCT 的系统做了比较，以期能够评估系统的总体性能。

测试平台四个节点组成：中间件节点和三个资源节点。中间件节点上安装了并部署了虚拟机调度及认证等服务，其硬件配置如下：Intel Xeon2 路 4 核 CPU，单核的主频为 1.60GHz，每一个核的高速缓存容量为 4MB，硬盘容量 80GB，物理内存 4GB。资源节点的 Xen 虚拟机监控器采用 TCT 机制，具有相同的硬件配置，具体如下：主频为 2.33GHz 的 Intel 双核处理器、2MB 二级缓存、2GB 内存和 80GB 7200RPM。网络出口带宽为 100Mbps。

3.5.1 功能测试

云计算是一种继网格计算后兴起的分布式计算模式[67][68]，由平台即服务、虚拟化、效用计算和软件即服务等概念和技术混合发展而来，通过高速网络和被称为“云”的资源池为用户提供按需的、经济的和易扩展的计算、存储和软件服务。这些庞大的资源池一般是集成了大量计算、存储和数据服务器的数据中心。云计算通过集中式的管理和资源的高利用率使得用户可以享受低成本和高可靠性的计算和存储服务。

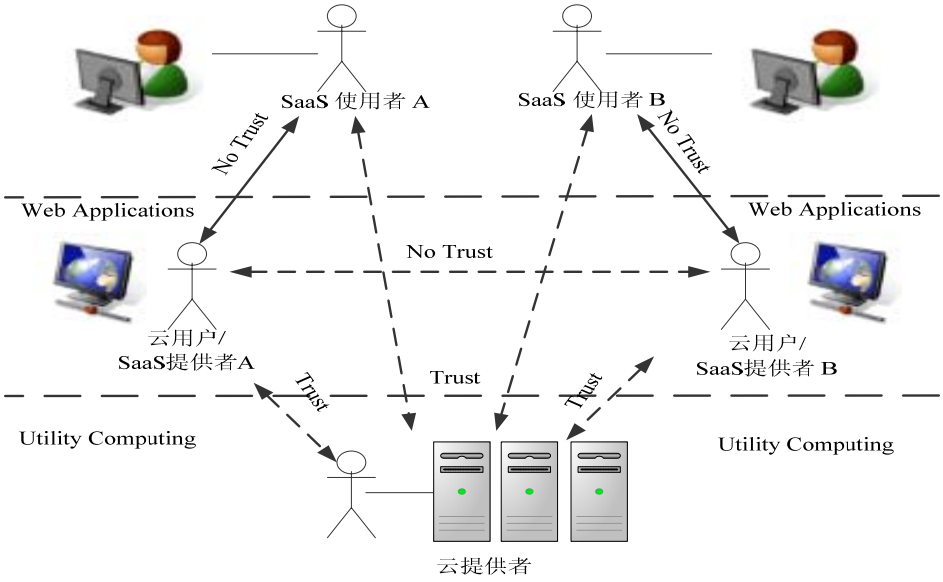

图 3.7 云计算中的信任关系

如图 3.7 所示，云计算中不同软件栈间的信任关系带来新的安全风险[32][70][71]。例如，在 Berkeley 的云模型中[66]，存在着三种不同的软件栈拥有者或是使用者：（1）云提供者--资源的提供者，拥有数据中心的硬件和软件资源；（2）云用户/ SaaS 提供者--租用云提供者的资源为公众提供 SaaS 服务；(3) SaaS 使用者。通常情况下这些不同的软件授权（Authorities）并不互相信任。例如热更新可能给 SaaS 提供者/云用户提供了一个获取 SaaS 使用者秘密的后门的能力。

现有的安全技术可以应用于云计算中来保护其安全。然而，这些技术提供的保护能力对云计算中的不同软件栈是不对称的。例如传统的入侵检测和病毒防护技术可以用于防止恶意的云用户或是 SaaS 使用者攻击云平台。云提供者同样可以使用虚

拟化技术为不同的云用户提供隔离的运行环境。然而对于上层的软件使用者，现有的技术不能防止下层的软件授权利用其作为下层软件栈提供者的优势损害上层用户的利益。TCT 可以依据 SaaS 使用者定义的信任集，动态地检查 SaaS 提供者提供的 SaaS 服务的完整性，并在 SaaS 使用者期望的环境完整性遭到破坏时，保护 SaaS 使用者的敏感数据。

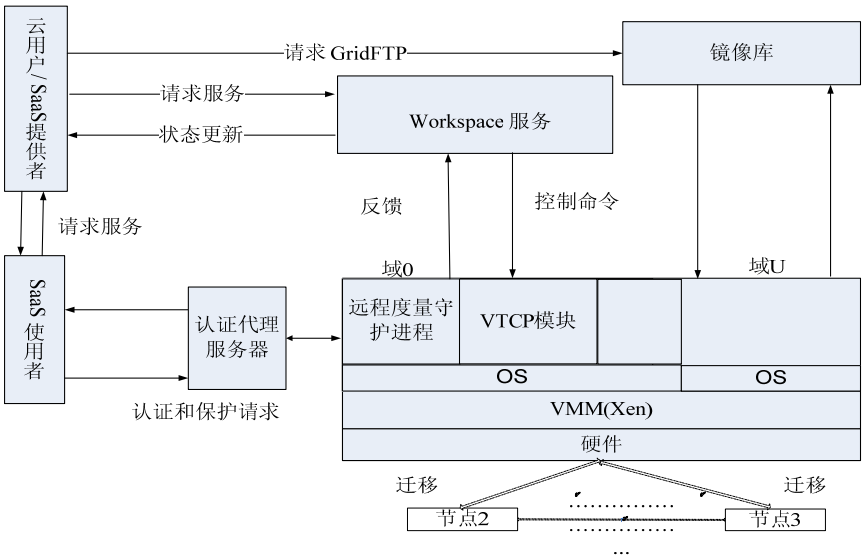

图 3.8 实现 TCT 的 Nimbus 架构

Nimbus 是一套开源的工具集，可以构建类似亚马逊架构的云计算解决方案。为了支持 TCT 的设计，在 Nimbus 云的每一个资源节点中需要采用 3.3 节中描述的实现技术修改 Xen 虚拟机监控器，并需要在每个节点的 dom0 中实现一个远程度量守护进程来报告每个虚拟机的度量信息。如图 3.8 所示，一个 SaaS 提供者在 Nimbus 的云平台上通过虚拟机镜像部署其服务，在本演示系统中，SaaS 提供者租用云资源仅提供一个简单的 FTP 服务。云计算的资源节点不应该暴露给 SaaS 使用者，因此在该演示系统中还实现了一个认证代理服务器（ADS）来作为 SaaS 使用者度量相应 VM 的通信桥梁。SaaS 使用者可以发送包含相应 FTP 服务 IP 地址的度量请求给 ADS，ADS 同相应资源节点中的远程度量守护进程通信，返回提供 FTP 服务的相应虚拟域中操作系统内核镜像和其他加载进程的度量列表给 SaaS 使用者。

SaaS 使用者验证该 FTP 服务的度量列表。如果其符合 SaaS 使用者的完整性需求，则可以对该度量列表进行签名并作为其可信集授权给 TCT 来保护其敏感数据。

SaaS 使用者需要将其保护数据的目录信息和经过签名的可信集提交给 ADS，由 ADS 通知相应的节点来监控 FTP 服务的完整性变化。如图 3.9 所示，具有 TCT 功能的 Xen 虚拟机监控器监控着 FTP 服务的完整性变化。当位于 SaaS 使用者可信集中的文件 /etc/profile 的完整性同其在可信集中的期望值不匹配时，Xen 虚拟机监控器检测到了这种变化，并对 SaaS 使用者的敏感数据进行保护保护（如图 3.10），即使拥有 Root 权限的 SaaS 的提供者也不能访问 SaaS 使用者的敏感数据。

```
[root@localhost disk_monitor_edit5]# ./monitor
**********env_check(dom_id)**********

**********parse_config(dom_conf)**********
Image :/root/guest/vm-ubuntu.img
ea24521cbaf6febb9f0f06349a986508  /etc/init.d/rc
e3756487011471f7753d5d94ce4b6af4  /etc/init.d/rc.local
6687b5585f865da7e7875b1d9cfff4a0  /etc/passwd
b59ea6ac3a1ad8c0527ec94f73bafca0  /etc/profile
```

图 3.9　监控 FTP 服务完整性变化

在具有 TCT 架构的云计算环境中，SaaS 用户的敏感数据同其期望的环境完整性绑定在一起，执行环境是否是值得信赖的则取决于 SaaS 用户自己的判断。例如，如果 SaaS 用户需要 FTP 服务为其上传的数据提供额外的加密服务，则需要自己去选择一个能满足其需求的 FTP 服务。TCT 可以通过度量 FTP 服务的配置完整性和将 SaaS 用户敏感数据同其配置完整性相绑定的方式确保 SaaS 用户选择的 FTP 服务不能欺骗 SaaS 用户。

```
[root@localhost-120 root]# ls -l
ls: cannot access protect.txt: Input/output error
total 0
-????????? ? ? ? ?            ? protect.txt
[root@localhost-120 root]# rm -rf protect.txt
rm: cannot remove `protect.txt': Input/output
error
[root@localhost-120 root]#
```

图 3.10 保护 SaaS 使用者敏感数据

3.5.2 性能

TCT 时间开销主要来自于虚拟机监控器对操作系统磁盘 I/O 和系统调用的语意解析和控制，本节对这两个部分中的时间开销进行测量，并同未采用 TCT 的系统进行了对比，从而评估 TCT 对系统的性能影响。

表 3.1 磁盘 I/O 的时间开销(ms)

	1KB	16KB	128KB	1MB	10MB
R_UNCHECK	7.7	17.8	107.8	862.4	10171.4
R_CHECK	8.7	21.8	139.6	864.2	12731.4
M_HASH	8.7	22.1	144.7	923.2	16322.2
W_UNCHECK	545.1	1173.2	5033.9	10580.2	16000.9
W_CHECK	547.2	1181.4	5097.9	10068.2	17623.9

表 3.1 是 TCT 磁盘 I/O 的测试结果。标签 CHECK 表示 TCT 原型系统的磁盘 I/O 操作所需的决策时间和操作时间。标签 UNCHECK 表示普通的 Xen 虚拟机监视器进行磁盘 I/O 操作所需的时间。标签 M_HASH 表示 TCT 原型系统在磁盘读操作的同时进行度量操作所需的时间。时间的获取是通过 Linux 的 time 命令。在上述不同的场景中，本文使用同一测试程序对从 1KB 到 10MB 的文件进行操作并测量其操作时间。

从表 3.1 中可以看出，TCT 架构对磁盘 I/O 的性能影响非常小。大部分的度量操作同磁盘 I/O 操作可以异步进行。DME 先进先出的队列设计使 TCT 架构中对磁盘性能影响比较大的操作可以同磁盘访问操作并行进行。

追踪系统调用是 TCT 架构中实现客户操作系统行为语义解析和 SaaS 用户敏感数据保护的关键技术。本文选用了一些基准测试程序来检测追踪系统调用对系统性能的影响。这些基准的测试程序包括 Linux Web 服务程序、数据库服务程序、 CPU 集中型的应用程序和数据解压缩程序（以 Linux 内核 inux-2.6.18.8.tar.gz 的镜像文件作为解压缩的目标文件，该内核镜像的大小为 58.6MB）。

图 3.11 是基准测试程序在普通 Xen 虚拟机监控器和 TCT 原型系统中性能的比较结果。从图中可以看出，TCT 架构对系统调用性能有一些影响，特别是对 Web 服务

基准测试程序的系统调用产生的延时比较大。造成这种情况的主要原因是 TCT 原型系统在实现中采用的通知机制。在现有的原型中，追踪系统调用是通过修改 Xen 虚拟机监控器的相关模块实现的，而决策模块则是位于 Dom0 的用户空间。所有系统调用的追踪结果都需要从 Xen 虚拟机监控器传递到决策模块。这种通知机制会造成很大的延时。因此，如果将决策模块实现到 Xen 虚拟机监控器中将会很大程度上降低对系统调用的延时。同时，从图中也可以看出，现有的实现方案中对 I/O 相关的系统调用性能影响不大。

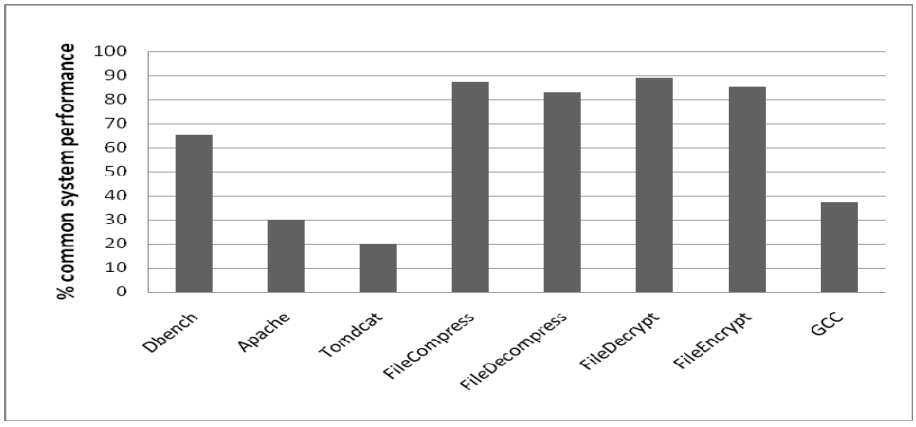

图 3.11 追踪系统调用的性能开销

3.6 小结

本章给出了基于虚拟机架构透明信任链机制的设计和实现，TCT 提供了一种对操作系统透明的信任链构建方式，可以将 TCG 方式的信任链扩展到应用程序层，同时 TCT 允许依赖方通过哈希值指定其信任的计算环境，并将依赖方的敏感数据同其期望的环境完整性绑定到一起。TCT 可以确保依赖方指定的敏感数据只能在其信任的环境中被访问，并当该计算环境遭到破坏后保护依赖方的敏感数据。试验结果表明了 TCT 架构可以应用于类似于网格和云计算等多重独立软件授权环境下来保护上层用户的敏感数据和系统的完整性，并且其性能开销是可以接受的。

4 基于轻量级虚拟化的动态可信执行环境构建机制

本章提出一种基于轻量级虚拟化的动态可信执行环境构建机制—Cherub。基于第二章的信任链理论，Cherub 构造从动态可信度量根到轻量级虚拟机监控器（LVMM）的动态信任链，并对目标程序进行细粒度的保护。同第三章 TCT 的设计目的不同，Cherub 主要用于提升桌面系统的安全性。

4.1 研究背景

在当今的信息化社会里，基于商用操作系统的桌面 PC 广泛地用于家庭、商业、政府甚至是军队。这些系统用于处理从个人口令密钥到关系社会安全的各种敏感数据，然而商用操作系统不能满足这些应用对安全性需求。操作系统作为可信基不仅包括操作系统内核同时也包括设备驱动和其他的一些同应用程序运行在同一特权级的系统服务程序，这使其通常包含大量的代码和易被攻击的各种漏洞。因此，商用操作系统不能满足高安全性应用的需求。

有很多机制用于来增强商用操作系统的安全，其中包括：基于 TCG 的可信启动机制来确保操作系统的完整性，在启动时度量整个操作系统，并将度量值提供给远端的验证者[38][42][43]，或是以操作系统作为可信基，来扩展 TCG 的信任链到应用程序层[39][40][41]。然而这些方式都以操作系统为可信基，如上文所述，操作系统作为可信基存在诸多安全隐患，即使可以确保其完整性，也不能为上层应用提供足够的安全支持；使用安全协处理器[31]或是具有安全功能的扩展处理器[44]，这些解决方案要求改变平台的架构和资源管理方式，对于商用操作系统并不适用。

虚拟机架构能在单一的物理平台上提供多个相互隔离的虚拟域，近年来，使用虚拟机监控器提供安全的运行环境成为一种流行方式[73][77]。现有的方法依据其提供的隔离粒度可分为两类：一类为高安全要求的程序提供一个单独的隔离虚拟域；另一类则为高安全需求的程序提供在原操作系统内的隔离运行区域。然而，这些安全架构都是基于通用目的虚拟机监控软件。通用目的的虚拟机监控软件是为服务器端的资源聚合和管理而设计的，因此存在诸多问题：1）虚拟化带来较大的性能开销，

这使用户在不需要高安全性的环境时，也需要承受虚拟化带来的开销。特别是对于客户端，大部分时间用户并不运行高安全性的应用；2）虚拟机为了更有效地满足服务端应用需求，自身代码量急剧膨胀起来[78]。 例如，Xen 虚拟机监控软件即使不包括特权管理域也有近 50,000 行代码。日益增大的代码量使虚拟机监控软件不可避免存在易被攻击的漏洞；3）现有方案缺乏一个有效的方式建立信任链来证明虚拟机作为可信基是可信赖的；4）对于虚拟机粒度的隔离，往往要改变应用程序的使用方式或是操作系统的资源管理方式而增加额外的性能开销。

为了解决上述问题，本章提出一种基于轻量级虚拟化的动态可信执行环境构建机制－Cherub。同传统的虚拟化方法不同，Cherub 利用动态可信度量根和硬件虚拟化技术，依据目标程序的保护需要在运行的操作系统中动态地插入轻量级的虚拟机监控器。轻量级的虚拟机监控器提供隔离的内存空间（PEX），将目标程序或其部分代码和数据的内存空间同操作系统的其他程序的内存空间相隔离。Cherub 通过控制对 PEX 中代码和数据的访问来实施灵活的保护策略。Cherub 可以依据事先定义的保护策略度量、加密、解密 PEX 中被保护的代码和数据。如果没有合法的访问许可，即使具有高优先级的内核代码也只能从 PEX 中访问到密文。同时，Cherub 的轻量级虚拟化设计能为目标程序提供透明和细粒度的进程保护，而不显著的降低系统性能。

4.2 硬件虚拟化和动态可信度量根

4.2.1 硬件虚拟化技术

硬件厂商在硬件的层次上提供了很多新的特性来简化软件层次的虚拟化实现。Intel VT[7][8]和 AMD 的 AMD-V[9]都引入了两个新的处理器状态（根态和非根态）及相关的扩展指令。两者的原理相似，但实现细节不同，本文主要以 Intel 架构为例来介绍硬件虚拟化。

在 Intel VT 中，引入的新的处理器操作被称为 VMX (Virtual Machine Extensions)。VMX 使虚拟机监控软件（VMM）可以运行于比特权级 0 更低的根态（VMX Root Mode），而客户机则运行在非根态（VMX Non-Root Mode）。处理器处于 VMX 根

态时的功能和权限就像没有 VMX 时一样，只是引进了一些支持 VMX 的指令，而处于非根态的处理器行为则受到了限制。

虚拟机的运行环境通过一个控制结构体进行设置，这个结构体同具体的处理器架构相关，在 Intel VT 中为 VMCS（Virtual Machine Control Structure）。VMM 对处理器的管理通过设置 VMCS 来实现，处理器从 VMX 根态进入 VMX 非根态时，处理器状态保存在 VMCS 中，同时客户机状态从 VMCS 中装入。相反，当虚拟机退出时，即从 VMX 非根态进入 VMX 根态，客户机状态保存在 VMCS 中，而 VMM 在根态的处理器状态则从 VMCS 中装入。

处理器的控制权由虚拟机转移到 VMM 被称为 VMExit，而处理器的控制权由 VMM 转移到虚拟机中被称为 VMEntry。某些特定的指令、事件或状态会导致虚拟机 VMExit 到 VMM，虚拟机也可以通过 VMCall 指令主动地 VMExit 到 VMM 中，但是在大多数应用场景中，运行于支持全虚拟化的虚拟机监控软件之上的虚拟机感知不到 VMM 的存在。虚拟机运行于 VMX 非根态，其大部分访问硬件资源的操作都会导致 VMExit 而将处理器的控制权转交给 VMM，因此 VMM 可以控制虚拟机的行为确保虚拟机不会访问到其他的隔离区域。

4.2.2 动态可信度量根

可信计算组织 TCG 的可信计算规范中定义了静态可信度量根（Static Root of Trust for Measurement，SRTM）。SRTM 从平台加电开始，按照启动顺序度量加载的模块来建立可信环境。而动态可信度量根（Dynamic Root of Trust for Measurement，DRTM）不依赖于系统加电重启的过程，而是允许从系统从任何一个不被信任的状态作为测量的起点来建立信任链。动态可信度量根需要 CPU 的支持。Intel 和 AMD 分别推出了动态可信度量根技术，Intel TXT(Trusted eXecution Technology)[35] 和 AMD SVM(Secure Virtual Machine)[36]。Intel TXT 技术和 AMD SVM 技术在原理上是类似的，Cherub 的原型实现基于 Intel TXT 技术。

采用 Intel TXT 技术，当要动态度量一个执行环境（MLE），必须要有一个具有 0 环特权的程序，将以下两个模块载入内存：MLE 和认证模块（SINIT AC Module）。

认证模块同特定的芯片组相关，由芯片生产商提供。动态可信度量根由 Intel TXT 技术的扩展指令 GETSEC[SENTER]提供。当这条指令被调用时，处理器验证认证模块的合法性。如果验证模块通过了检查，该指令则将控制权交给认证模块来度量 MLE。度量的哈希值被保存在 TPM 的动态 PCR 中。

TCG 规范中定义了平台配置寄存器（PCR），从信任根开始，在将 CPU 的控制权移交到下一个模块前，都要先度量其完整性，度量的结果将保存到 PCR 中，由此建立信任链。为了支持动态可信度量根，在 TPM1.2 规范中定义了静态和动态 PCR。静态 PCR 在系统加电重启时被重新设置，而动态 PCR（PCR17-23）只能由动态可信度量根来重置为零，重新启动只能将动态 PCR 的值重置为-1，因此依赖方可以通过查看动态 PCR 的值来区分是由重新启动建立的信任链还是由动态可信度量根建立的信任链。

4.3 目标与挑战

4.3.1 威胁模型

假定基于软件的攻击可以危害到操作系统并可能完全控制操作系统。在这种情况下，攻击者可以执行用户态和内核态任意的处理器指令。例如攻击者可以通过特权指令访问到处理器的寄存器，修改页表甚至使用其指定的参数调用 GETSEC[SENTER]指令。

对于硬件级别的攻击，假定攻击者只能在系统关闭电源后通过物理接触的方式进行攻击。攻击者可以使用诸如打开机箱、重启系统或是使用硬件调试器等简单的攻击方式，也可以利用具有 DMA 能力的硬件进行攻击，例如 PCI 总线上的以太网卡等。但是这里假定攻击者不能发起例如监控处理器与内存高速总线之类的复杂攻击[18]。

4.3.2 设计目标

通用虚拟机监控软件需要将平台资源安全地隔离为多个不同的虚拟域，并控制每个虚拟机中操作系统的硬件访问请求。Cherub 架构只提供用于安全目的轻量级虚

拟化可信基，而并不需要模拟整个硬件环境来隔离多个虚拟域。因此在大多数时候，Cherub 并不干预客户机的运行，允许客户操作系统直接访问大部分的硬件资源，除非是在为了保护它自己或是为了实现它的安全目标。这意味着 Cherub 和客户机共享大部分资源，这也使 Cherub 的轻量级虚拟化设计对系统性能的影响非常小。总的来说，Cherub 要满足如下设计目标：

小的可信基 作为平台的可信基，Cherub 的代码量应该越少越好，但是必须能实现其安全目标和保护它自己不受恶意的攻击。

尽量小的性能影响 虚拟机架构用于增强操作系统安全时，面临的一个很大的问题是虚拟化带来的性能开销。Cherub 需要解决传统虚拟机架构不能动态引导以及虚拟化开销大的问题。

不需要修改客户操作系统 这意味着 Cherub 功能设计不能依赖于修改操作系统。修改操作系统可能使其更加容易遭受攻击，并且对操作系统的任何修改都要重新编译操作系统内核，这在大多数情况下是不可行的。同时，采用虚拟化技术来加强操作系统安全之所以非常有吸引力，其中一个很重要的原因是虚拟化技术可以在保持对操作系统透明的同时增强其安全性。修改操作系统将部分安全模块实现到客户操作系统中将有损虚拟化技术在增强操作系统安全方面的优势。

不需要修改应用程序 Cherub 应该能够保护很大范围内的遗留程序，这意味着不能修改现有应用程序的编程模型。

细粒度的隔离保护环境 Cherub 能提供细粒度的隔离运行环境，可以依据程序使用者或是开发者的需求对目标程序的敏感数据和对敏感数据的操作进行保护。

Cherub 遇到的首要挑战是如何保护它自己。Cherub 是在操作系统运行中加载的，在其被加载以前，Cherub 只是普通的内核模块。恶意的操作系统可以在其加载前破坏 Cherub 的完整性，因此如何确保 Cherub 是值得信任的。其次当 Cherub 被加载后，Cherub 同客户操作系统共享大多数资源，如何将 Cherub 同客户操作系统隔离开，以避免恶意操作系统的攻击。第三，和 Cherub 类似，微软的 NGSCB[28]为进程提供细粒度的隔离运行环境，允许进程将其代码和数据的一部分保护起来，NGSCB 通过提供新的调用接口，让程序主动调用这个接口来告知 NGSCB 保护的部分。Cherub 不

改变现有的编程模式，那么如何通知 Cherub 目标程序需要保护的代码数据信息。第四，Cherub 要通过一个尽量小的可信基来提供对目标程序的保护，这意味着 Cherub 的设计要在功能、灵活性、性能和小的可信基之间找到平衡。

4.4 Cherub 架构设计

4.4.1 Cherub 架构概述

图 4.1 展示了 Cherub 的设计架构。Cherub 利用动态可信度量根（DRTM）和硬件虚拟化技术在运行的操作系统下加载一个可信的轻量级虚拟机监控器（LVMM）。由 LVMM 为目标程序创建和维护被保护的执行环境（PEX），并通过监控对位于 PEX 中的目标程序的内存页面的访问。LVMM 确保只能在 PEX 中才能对敏感数据进行操作。Cherub 通过由目标程序开发者或是发行商提供的保护策略来决定目标程序的保护部分。

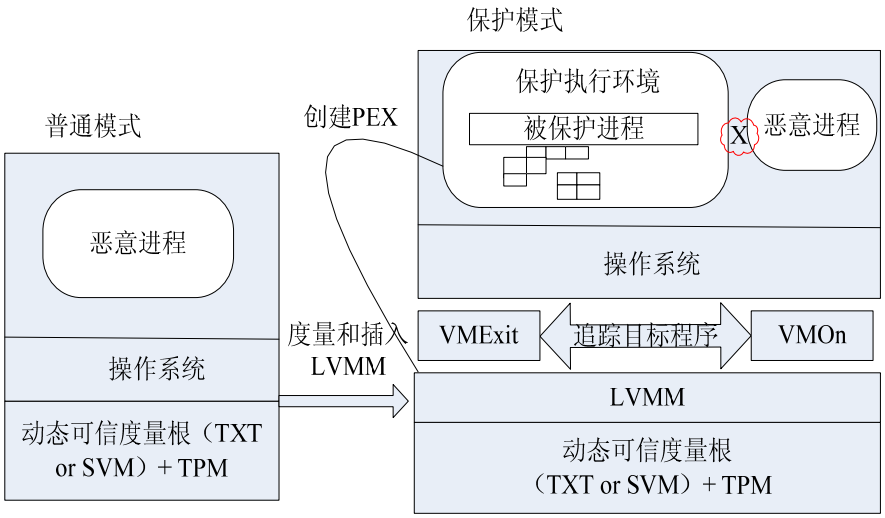

图 4.1 Cherub 体系结构

Cherub 的优势源自于其轻量级的虚拟化设计，动态信任链的构建，和通过 PEX 实现隔离计算能力，因此 Cherub 要实现四个方面的功能：

运行时加载 LVMM Cherub 利用硬件虚拟化提供的扩展指令在运行的操作系统中载入 LVMM，并将该操作系统迁移到虚拟机中。

LVMM 的可信启动和被保护代码及数据的完整性 Cherub 使用动态可信度量根（DRTM）技术可以保证轻量虚拟机监控器通过一种可信的方式启动。由 TPM 提供轻量虚拟机的完整性证明，轻量虚拟机可以度量 PEX 中被保护代码和数据的完整性。

运行态 LVMM 完整性保护 运行阶段， Cherub 通过影子页表机制将 LVMM 和安全功能模块的页表从操作系统中隐藏起来，并通过 I/O 硬件虚拟化技术保护它们免受恶意 DMA 的攻击[8][9]。

DRTM 和 TPM 可以保证 LVMM 及安全功能模块从可信根加载t_0时刻到操作系统恢复t_n的完整性，从而使上述实体不依赖于平台在时刻t_0前载入的实体。运行态 LVMM 及安全功能模块的完整性保护，可以使这些模块实体在t_n时刻后不依赖于操作系统和其他平台实体。从而构建出符合本文 2.2 中模型的动态信任链。

被保护执行环境（PEX）Cherub 在 LVMM 影子页表中标记目标程序的被保护代码和数据，因此 LVMM 可以检查对这些内存页访问的合法性。同时 LVMM 在处理器的控制权移交给 PEX 外的代码时，加密 PEX 中代码和数据所在的页。当控制权返回 PEX 时，经过 PEX 合法的入口点检查后，LVMM 解密这些页面。LVMM 同样依赖 I/O 硬件虚拟化保护 PEX 免受恶意 DMA 的攻击。

4.4.2 动态载入 LVMM

Cherub 的一个设计原则是 LVMM 并不永驻内存，而是依据目标程序的保护需求动态加载，并且加载和卸载 LVMM 就像加载和卸载操作系统的可加载内核模块一样简单。Cherub 利用硬件虚拟化技术实现 LVMM 的动态载入并将运行中的操作系统迁移到虚拟机中。

Cherub 首先需要对控制虚拟机运行环境的结构体进行设置。这个结构体中定义了虚拟机的状态包括例如段寄存器、CR3 寄存器和指令寄存器等寄存器值。Cherub 克隆几乎所有 CPU 状态到这个结构体中，而通用寄存器通常不包含在这个结构体。在 Cherub 加载 LVMM 以前，操作系统控制所有的硬件资源，当 LVMM 加载后，LVMM 运行于 VMX 根态而原有操作系统运行于 VMX 非根态。Cherub 通过设置这

个结构体，使操作系统在 VMX 非根态几乎运行于和其迁移前相同的环境中，从而使其感觉不到运行于虚拟机中。

运行于 VMX 非根态的操作系统会因为很多原因陷入 VMX 根态（例如执行某些特权指令或是需要处理中断等），把控制权转交到 LVMM，Cherub 设置一些处理函数来处理这些陷入事件。Cherub 依据其设计目标来设置这些处理函数，当设置完毕后，Cherub 提供一个客户机的执行入口点并执行特定的虚拟化扩展指令（VMRUN for AMD-V and VMLAUNCH for VT-x）将控制权交给操作系统，操作系统从这个入口点继续执行。

当LVMM提供的保护功能不再需要时，例如当所有的目标程序都已经结束运行，Cherub 可以卸载 LVMM，操作系统将重新控制所有的系统资源。动态的移除和加载 LVMM 使 Cherb 具有其他基于通用虚拟机监控软件的安全架构所不能比拟的灵活性和性能优势。

4.4.3 信任 LVMM

4.4.3.1 可信启动

正如前文提到的那样，Cherub 在运行的操作系统中加载 LVMM 就如同加载内核驱动一般。然而操作系统可能被攻击者侵害，因此 LVMM 应该在其加载前被度量和验证。问题是谁来度量 LVMM 才是值得信赖的。Intel TXT 和 AMD SVM 都提供了动态可信度量根技术（DRTM），该技术可以在任何时间度量一段载入内存的代码而不需要依赖平台加电重启。Cherub 利用该技术在需要的时候度量 LVMM，实现 LVMM 的可信启动。

Cherub 首先要将 LVMM 代码和数据定位到指定的内存区域，然后调用 DRTM 指令，该指令度量这个指定的内存区域，存储度量的结果到 TPM 中，接下来将控制权移交给度量内存中 Cherub 指定的初始化程序，由该程序完成系统的配置工作。依赖方可以通过存储在 TPM中的度量值来确认 LVMM 的完整性。

4.4.3.2 运行态 LVMM 的完整性保护

Cherub 可以使用与通用虚拟机监控器类似机制，使用私有地址空间来简化 LVMM 的内存保护，即 LVMM 的物理内存页不被映射到客户机的页表中。但是 Cherub 的一个设计目标是使用尽量小的 TCB 来提供保护功能。为了这个目标，Cherub 的虚实地址映射和客户操作系统的虚实地址映射几乎相同。这种方式可以使 LVMM 非常方便地监控内存和追踪操作系统的行为而对性能的影响却非常小，并且极大减小了 LVMM 的代码量。然而同客户操作系统使用相同的虚实地址映射也为 LVMM 带来了潜在的安全风险。一旦攻击者控制了操作系统内核，便可以攻击 LVMM 和安全功能模块，例如通过将 LVMM 占用的物理地址映射到其他虚地址的方式对其进行读写。

Cherub 采用一种冗余页表机制来保护 LVMM 和安全模块。在 LVMM 加载以前，Cherub 的启动模块只是一个普通的 linux 内核模块，内核模块所需要的内存空间都需要向操作系统申请。启动模块向内核申请多于 LVMM 实际需要的内存，将多余的内存作为冗余页来保护 LVMM 和安全功能模块实际使用的页。当 LVMM 加载后，LVMM 运行于 VMX 根态，这使其拥有比客户操作系统更高的特权级可以干预客户操作系统的行为。同时硬件虚拟化技术使 LVMM 可以控制对 CR3 寄存器的操作，LVMM 为客户操作系统创建冗余影子页表。在影子页表中，LVMM 将其和安全模块实际占用的页都映射到冗余的物理页上，并使用和通用虚拟机监控器类似的影子页表机制[11][17]管理客户操作系统对页目录的访问。因此，当客户操作系统访问 LVMM 和安全功能模块的内存时，它只能访问到冗余的物理页。

这种机制可以有效地防止客户操作系统对 LVMM 内存页的访问，然而具有 DMA 能力的硬件设备却可以绕过冗余页的保护直接访问 LVMM 的物理内存。Cherub 需要依赖设备 I/O 虚拟化技术来解决这个问题[8][9]。通过设备 I/O 虚拟化可以使硬件来完成客户物理地址和真实机器地址的转换，通过设置 NODMA 表可以保护 LVMM 和安全功能模块不被 VMX 非根态下的设备访问。

4.4.4 保护执行环境

当 LVMM 载入后，LVMM 通过创建保护执行环境（PEX）来保护目标程序的代码和数据。LVMM 可以创建多个 PEX 为不同目标程序服务，为了描述上的需要，本文只选取一个 PEX 来解释其工作原理。如图 4.2 所示，Cherub 允许应用程序将其代码和数据分为被保护部分（可信部分）和不保护部分（非可信部分）。对 Cherub 来说，除了目标程序的被保护部分，客户机中其它的代码包括操作系统都被认为是不可信的部分（在不同 PEX 中的代码彼此间也是不可信任的）。LVMM 通过在影子页表中标记这些被保护的代码和数据所在的页面来区分其它的不可信部分。对于这些被保护的代码和数据，LVMM 在其首次被加载进内存时，度量它们并同已有的标准值进行比较来验证其完整性。这些标准值由目标程序的开发者或是提供者给出。在目标程序运行阶段，LVMM 追踪其运行的全过程并确保 PEX 中的数据只能被 PEX 中的代码访问，同时验证进入 PEX 入口点的合法性是否满足程序的开发者或是提供者预先给定的安全策略。

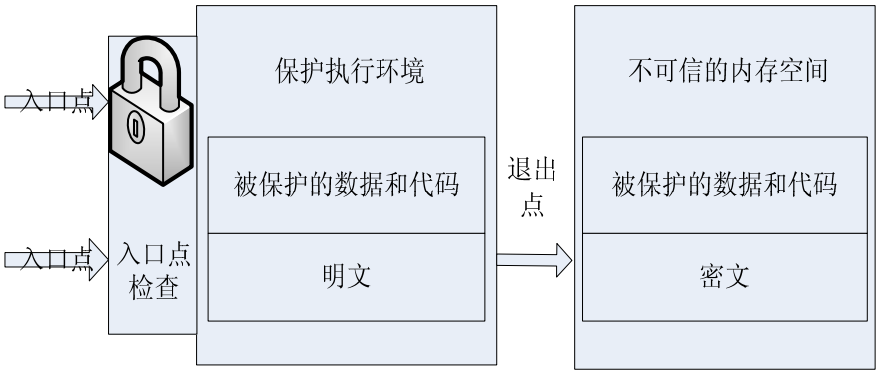

图 4.2 被保护执行环境

入口点和退出点 Cherub 允许不可信的代码通过预先定义的入口点调用位于 PEX 中的可信代码。初始的入口点都是由程序的开发者或是提供者定义的，在运行阶段，合法的入口点还包括一些临时入口点。退出点是因为在 PEX 中的代码需要将处理器的控制权交给 PEX 外的不可信代码，通常由以下几种原因造成：1)PEX 中的代码进行系统调用或是调用了不可信的函数；2）在 PEX 中的代码运行时，发生了中断；3）PEX 中的代码执行完毕，将处理器的控制权移交给不可信代码。前两种原

因将产生临时的入口点，因为调用或是中断处理完毕后，处理器的控制权还需要从该退出点返还给 PEX 中的代码。LVMM 将检查从这些入口点的合法性，从而确保目标程序的被保护代码依照事先定义的路径执行。

追踪进入和退出 PEX Cherub 需要一种机制，当处理器的控制权在 PEX 内外转换时可以产生 VMExit，从而陷入 LVMM 中，使其可以验证进入和退出 PEX 的合法性。LVMM 利用 X86 架构分页的硬件保护机制，在影子页表中通过设置某些标志位来实现期望的 VMExit。例如，对于由系统调用或是中断带来的退出点（临时入口点），进程将由用户态进入内核态，LVMM 可以通过设置内核页表的可执行位而使只有期望的内核代码才可以执行。当进入内核态时，因为相关的代码被设置为不可执行时，处理器将产生页错误而将控制权交给 LVMM。LVMM 记录下这些临时入口点的线性地址，并将这些入口点代码所在页的 user/supervisor 位设置为 0，这意味这些页只能在内核态才能被执行。因此当从系统调用或是中断处理例程执行完毕返回用户态时，又会产生页错误而将控制权交给 LVMM。这种机制可以使 LVMM 追踪进入和退出 PEX 的过程。

动态加解密 PEX 的内存页 恶意的内核可以通过将 PEX 中的页面映射到其它虚地址等方式旁路入口点的检查而访问 PEX 中的页面。因此 LVMM 在每一次进入和退出 PEX 时，对 PEX 中的页面采用动态加解密的方式进行保护。当退出 PEX 时，LVMM 加密 PEX 中所有的页面，而当进入 PEX 时，LVMM 首先检查入口点的合法性，如果通过了检查，LVMM 解密所有的页面。通过这种方式，被保护的代码只能在 PEX 中通过合法入口点检查后才能被执行，而被保护的数据则只能由被保护的代码访问。在 PEX 外的不可信代码（包括内核）看到只是密文。

度量表单 目标程序通过度量表单通知 Cherub 其被保护的代码和数据及这些被保护代码的执行策略。度量表单通常由程序的开发者或是提供者提供。度量表单中包括被保护数据和代码在内存中的重定位信息、其初始标准哈希值、入口点的偏移量等信息。

4.5 Cherub 的核心机制

Cherub 同操作系统和具体的硬件无关，但要求处理器支持硬件虚拟化和动态可信度量根扩展指令（这两个扩展已经成为 Intel 和 AMD 商用处理器的主流）。本文仅给出基于 Intel TXT 和 Linux 操作系统上的实现。

Cherub 原型包括三个主要模块：载入模块、启动模块和轻量级虚拟机监控器（LVMM）。Intel TXT 的动态可信度量根扩展指令 GETSEC[SENTER]只能在内核态（ring 0）执行。因此在启动 LVMM 前，需要利用 Linux 提供的可装载内核模块（Loadable Kernel Module，LKM）机制，加载一个启动模块来执行这些特权指令。启动模块还需要配置 LVMM 镜像的内存映射，并在 LVMM 加载后作为用户和 LVMM 通信桥梁。目标程序和其度量表单文件需要向载入模块注册，才能受到 LVMM 的保护。图 4.3 展示了 Cherub 各模块的启动过程。

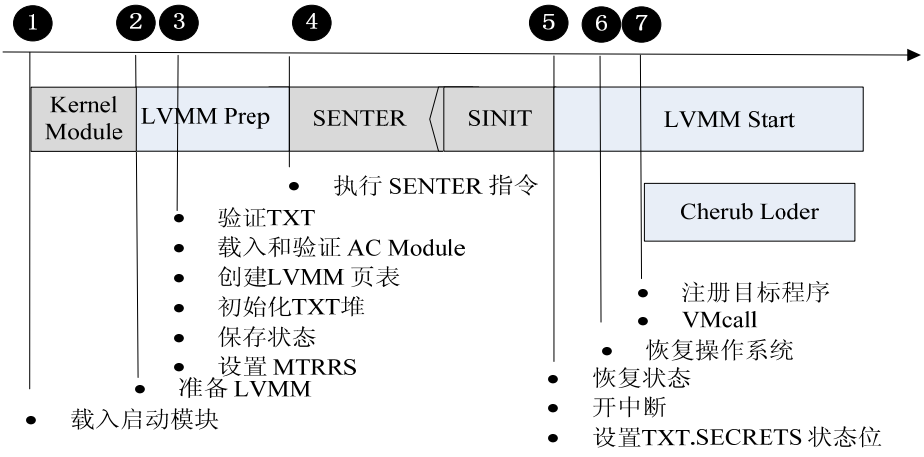

图 4.3 Cherub 模块的启动过程

4.5.1 LVMM 的载入过程

4.5.1.1 LVMM 内存镜像和 TXT 堆

在 LVMM 加载前，启动模块首先对系统硬件进行检查，包括处理器的 SMX、VMX 扩展、TPM 芯片状态等情况。检查通过后，启动模块将 AC Module 载入内存，并为 LVMM 分配足够的运行空间，将 LVMM 镜像文件读入分配的空间中。Intel TXT

并不要求被度量的执行环境（MLE）的物理地址必须是连续的，为了减少 LVMM 的代码量和性能开销，启动模块将 LVMM 的物理地址依照线性地址由低地址到高地址进行存放。如图 4.4 所示，LVMM 内存映像包括四个部分：代码段，数据段，堆栈段和临时数据区，但是只有 LVMM 的代码段、数据段和堆栈段作为被度量执行环境（MLE）。同时，启动模块为 LVMM 的代码段和数据段创建一个页表，存放在 LVMM 的代码段前。

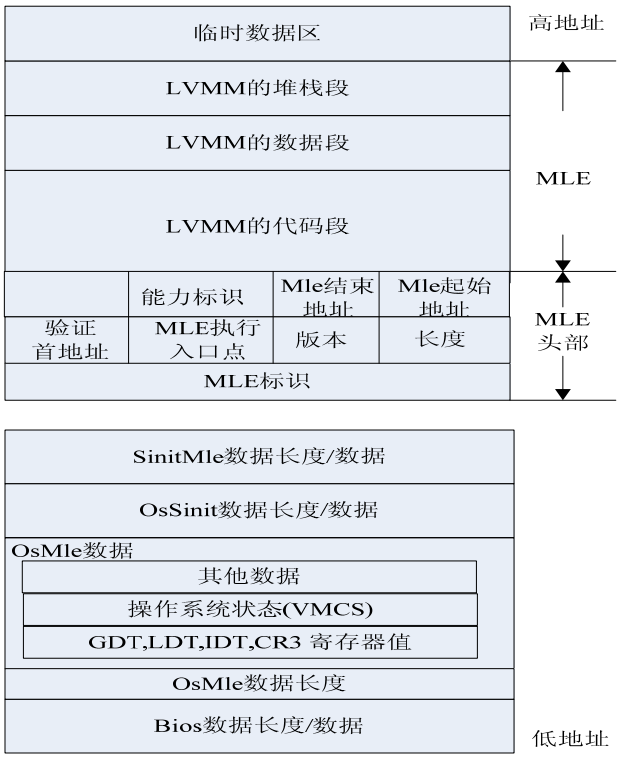

图 4.4 LVMM 内存映像及 TXT 堆

启动模块还需要保存操作系统的当前状态，用于 LVMM 启动后恢复操作系统。支持 Intel TXT 技术的处理器大多都拥有多个内核，这使恢复操作系统变得异常复杂。为了减小开销和 LVMM 的代码量，在执行 GETSEC[SENTER]前，启动模块利用 Linux 内核（版本 2.6.19 后）的 CPU 内核热拔插（Hotplug）技术休眠除了引导处理器(Boot-strap Processor)外的其他内核。这样只需要保存引导处理器的当前状态，这些状态信息被启动模块保存在 Intel TXT 堆中。Intel TXT 堆是由 BIOS 设置的一段连续的格式化区域。除了系统的状态信息，启动模块还需要将 MLE 的页目录物理基址

和大小信息填入 TXT 堆的 OsSinit 数据段。LVMM 在启动后并不使用 MLE 的页表，而是使用和操作系统相同的虚地址和物理地址映射。因此启动模块还要将 GDT（Global Descriptor Table）、LDT（Local Descriptor Table）、IDT（Interrupt Descriptor Table）和当前的 CR3 值保存到 TXT 堆的 OsMle 数据段。

4.5.1.2 度量 LVMM

完成 LVMM 的镜像载入和 TXT 堆初始化后，启动模块调用 GETSEC[SENTER]指令启动度量过程，引导处理器度量认证模块并同存放在认证模块头部的由芯片提供商签名的该模块的哈希值相比较，由此来检查验证模块的完整性和合法性。如果通过验证，验证模块的哈希值将会扩展到 PCR17 中。接下来，处理器的控制权将交给认证模块，认证模块将验证系统的配置，保护 MLE 的内存空间不被非法的 DMA 设备访问，度量 MLE，并将哈希值扩展到 PCR18 中。认证模块操作完成后，执行 GETSEC[EXITAC]指令将控制权交给 MLE，将从在 MLE 头部指定的 MLE 执行入口点处继续执行下条指令。

4.5.1.3 LVMM 初始化

Cherub 在 MLE 头部指定的执行入口点指向 LVMM 的初始化代码。该段代码首先检验页表是否正确地映射了 LVMM 占用的内存空间。接下来根据存储在 TXT 堆中内容初始化 LVMM 的系统环境，因为 LVMM 和操作系统使用相同的虚实地址映射，LVMM 的运行环境和操作系统类似。它们使用相同的 CR3、GDT、IDT 和 LDT。

初始化代码设置 CR4 控制寄存器中相应的状态位来启动 VMX 模式，并在以 4KByte 对齐的内存上创建 VMXON 区域，然后传递 VMXON 区域的物理地址，执行 VMXON 指令。

VMXON 指令执行后，处理器处于 VM-Root 状态。处理器的控制权移交给 LVMM，LVMM 将在 VMON 区域上创建 VMCS。 VMCS 是一个数据结构用于控制 VMX 非根态和根态的转换。VMCS 有六个数据段：客户状态区（GuestState Area）和宿主状态区（HostState Area）分别用于存放进入和退出 VM 时处理状态，当处理器在 VMX 非根态和根态转换时，相应的处理器状态便由这两个区域载入。LVMM

根据存放在 TXT 堆中的信息来初始化 VMCS 的客户机状态（GuestState），包括从根态切换到非根态时需要加载的处理器上下文等。

虚拟机执行控制区（VMExecution Control Fields）和虚拟机退出控制区(VMExit Control Fields)控制在 VMX 非根操作时处理器的行为。它们决定了虚拟机由非根态进入根态原因。虚拟机入口控制区（VMEntry Control Fields ）控制 VM 进入。虚拟机退出原因区（VMExit Information Fields）描述 VM 退出的原因和特性。除了实现自身的安全目标外，LVMM 不需要对这些区域进行以隔离资源为目的的填充和处理。

接下来，LVMM 为客户操作系统创建影子页表，和普通虚拟机监控软件类似，LVMM 为客户操作系统管理着并行的页表。客户操作系统维护的页表称为客户页表(GPTs)。LVMM 维护着和 GPTs 并行的页表被称为影子页表(SPTs)。实际的虚拟地址到机器物理地址的转换是由影子页表来完成的。在影子页表中，除了 LVMM 和安全功能模块占用的内存空间外，虚实地址的映射关系和 OS 被迁移前是一致的。

对于 LVMM 占用的内存空间，Cherub 使用 4.4.2 中描述的冗余页机制。同时 LVMM 使用类似 Xen 的内存管理机制追踪客户页表的修改并同步影子页表。因为影子页表和客户页表的大部分映射一致，Cherub 的同步算法同 Xen 相比要简单的多。影子页表机制依然是虚拟化开销中比较大的部分，现有的硬件虚拟化技术例如 AMD 的嵌套分页（nested paging）[9]和 Intel VT-x[8] 可以显著地减小影子页表的性能开销。

最后 LVMM 设置 DMA 保护范围，并执行 VMLAUNCH 指令， 原有的操作系统被透明地迁移到 LVMM 的虚拟机中继续运行。

4.5.2 Cherub 的通信方式

LVMM 加载后，其运行于 VMX 根态，Cherub 需要一种机制将目标程序的度量表单和用户的控制参数从用户空间传递给位于 VMX 根态的 LVMM，同时需要将一些反馈信息（例如度量日志）传递回用户空间。

如图 4.5 所示，载入模块提供用户接口。当目标程序通过载入模块加载时，目标程序磁盘镜像的路径、度量表单的存储路径和控制参数由载入模块通过两个虚拟文件系统接口传递给启动模块。启动模块从磁盘中将度量表单读入内存，并将度量表

单的内存镜像放入地址连续的页（如果度量表单的内容超过一个内存页），然后通过保存在度量表单头部的由目标程序提供者或是开发者签名的表单文件哈希值验证度量表单的完整性。接下来，启动模块调用 VMCALL 指令将目标程序的名称和度量表单存放页的入口线性地址传递给 LVMM。因为 LVMM 和客户操作系统使用相同的虚实地址映射，LVMM 通过该线性地址度量该表单，并将其哈希值存入 TPM 的 PCR18 中，然后将度量表单的内容拷贝到 LVMM 的数据段，最后将目标程序的名称加入到被保护文件列表中。

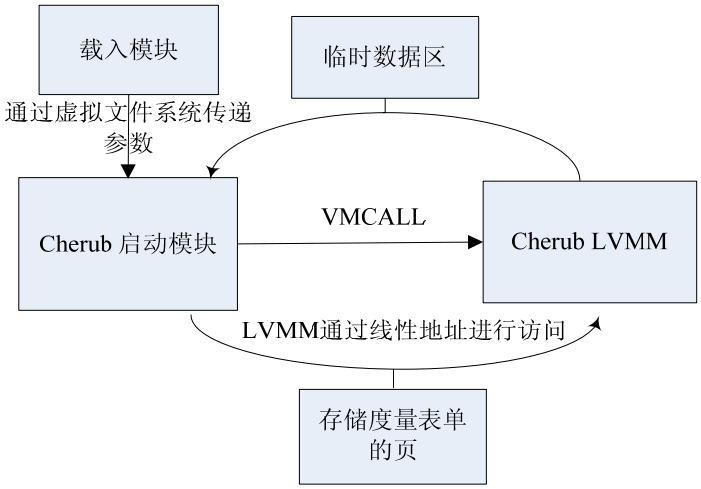

图 4.5 Cherub 的通信方式

LVMM 将度量日志等反馈信息填充到临时数据区。临时数据区占用的内存页在启动模块准备 LVMM 内存镜像时被固定下来。LVMM 在临时数据区设置两个标志位指示对临时数据区的填充和读取，启动模块轮询其中的 Finish 标志位，当发现该标志位被置位，则将度量日志等反馈信息写入用户指定的目录，并将 Read 标志位置位，通知 LVMM 启动模块已经读取了上述信息。

4.5.3 目标程序保护

4.5.3.1 追踪目标程序

运行阶段，LVMM 在进程上下文切换时追踪目标程序。当进程上下文切换时，LVMM 获取控制权同时使用 VMI 机制[79]获取即将运行的进程 ID 或是名称。如果获

取的进程信息同向 Cherub 载入模块注册的进程信息相同，则 LVMM 依据该程序的度量表单信息对其进行保护。

LVMM 在 CR3 切换时，从 Linux 内核的 task_struct 结构体中获取即将运行的进程信息。当 VMExit 发生时，LVMM 从 VMCS 的 VM_EXIT_REASON 中可以获取 VMExit 的原因。如果其值为 EXIT_REASON_CR_ACCESS，LVMM 可以判断出 VMExit 的原因为 CR3 切换。接下来，LVMM 可以通过屏蔽 ESP 寄存器的低 12 位获取 thread_info 基址。thread_info 的第一区域内存放着指向 task_struct 的指针，从而 LVMM 可以获取即将运行的进程信息。

4.5.3.2 度量表单

LVMM 使用目标程序度量表单中提供的信息对其进行保护，图 4.6 给出了度量表单的格式。度量表单由头部和正文两部分组成。头部包含由该表单提供者签名的表单正文段哈希值和提供者的公钥。正文段则进一步被分为多个小节，每个小节中包含该节的头部和正文。小节的头部指明该节的类型和长度，在本文的实现中只支持两种类型，标记为 0 表示该节是数据段，而标记为 1 则表示该节是代码段。

图 4.6 Cherub 度量表单的格式

每一小节的正文段包含目标程序中被保护代码和数据的以下信息：1）以页对齐的被保护部分的 SHA-1 值；2）重定位信息。当目标程序被载入内存，操作系统将重新定位磁盘镜像中的代码和数据。因此，为了能够保护这些代码和数据，Cherub 需要重定位的信息。3）外部访问标志。该标志定义某个代码页的访问属性。例如如果该页的外部属性页被设置为 1，则表示该页的代码可以从 PEX 的外部调用，即该页包含一个入口点。入口点不需要是页对齐的，但是入口点的相对偏移量必须给出。

4.5.3.3 双影子页表

LVMM 通过双影子页表的机制实现 PEX 的保护功能。当目标程序第一次准备运行或是目标程序发生页缺失时，LVMM 检查目标程序中是否有在度量表单中的页被载入内存。如果有，LVMM 为这些页创建保护影子页表。LVMM 依据度量表单中的信息，将这些被保护页表的线性地址在影子页表中的每一级映射拷贝到 PSPts 中。如图 4.7 所示，所有的被保护代码和数据所在的物理页和客户操作系统的内核页都同时被映射到保护影子页表和影子页表中，但是它们的状态位是不同的。在影子页表中所有被保护代码和数据所在的页的 user/supervisor 位被设置为 0 ，而在保护影子页表中所有的系统调用例程和中断服务例程被标记为不可执行，其它不可信的用户态代码所在页的 present 位被标记为 0 。最后 LVMM 克隆所有的堆栈段和其它数据段的页面映射到保护影子页表中。

被保护的数据和代码所在的页面依据度量表单将会被度量和验证。只有通过完整性检查的页面才会映射到保护影子页表中。如果它们的哈希值不匹配，Cherub 给出了一个选项，这些页面同样会被映射到保护影子页表中，但其度量值将记录到度量日志中。这个选项的参数可以通过前文所述的方式在该目标程序加载时指定。LVMM 使用一个随机产生的向量加密所有被映射到保护影子页表中的被保护数据和代码所在的页，并对密文进行哈希运算。加密的向量和产生的哈希值被 LVMM 保存起来用于将来进行解密。同时，LVMM 将所有在度量表单中的入口点线性地址保存在一个 entry_point_list 的结构体数组中。

当 CR3 指向 SPTs 时，意味着不可信的代码正在运行。此时如果不可信的代码将控制权移交给可信的代码，将会产生一个页错误，因为 SPTs 中被保护的代码所在

的页的 U/S 标志位为 0，这代表这些页只能在内核态才能被访问。页错误将导致 VMexit 到 LVMM 中。LVMM 可以将入口点的线性地址同保存在 entry_point_list 的结构体数组中的值进行比较来验证该入口点的合法性。如果是合法的入口点，LVMM 首先对加密的页面做哈希运算并同先前保存的值比较。如果不同，程序将终止运行，如果相同，LVMM 解密这些页面，并使 CR3 值指向 PSPts，程序将从入口点开始执行。

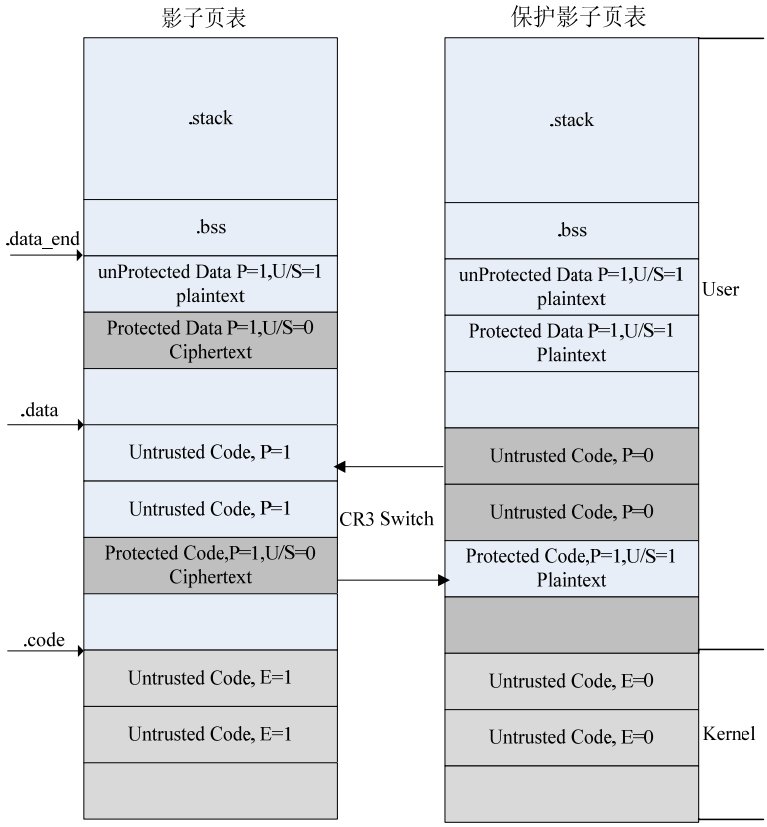

图 4.7 双影子页表设置

当被保护的代码执行完毕、调用不可信的代码（包括系统调用）或是发生中断，同样会产生一个页错误，因为这些不可信代码所在的页面 present 标志位为 0 或是被标记为不可执行。这将同样 VMexit 到 LVMM 中，LVMM 采用和这些页加入 PSPts 时相同的方式重新加密这些页，如果执行还需返回，LVMM 将这些临时入口点加入 entry_point_list 的结构体数组中，然后将 CR3 指向 SPTs。

4.5.3.4 优化加解密开销

加密和解密内存页面不可避免地带来性能的开销。Cherub 使用多副本和最小同步的机制[74]来最小化性能的开销。多副本机制通过保留同一个页的多个副本减少加解密操作需要处理的页面数量。而最小同步机制，进一步减小多个副本间的同步开销。具体来讲，在 LVMM 的内存中保留被保护代码和数据所在页面的明文副本，并对该加密页进行度量保留其哈希值。当一个加密的页需要解密时，对其做同样的度量如果哈希值相同，意味着该页没有被修改过，则直接使用其明文进行替换。

4.6 安全性分析及性能评测

4.6.1 安全性分析

在不同上下文中，对目标程序被保护代码和数据所在的内存页进行动态加解密，使 Cherub 能够阻止不可信的代码访问被保护的代码和数据，所有没有经过授权的访问，即使是具有高特权的内核代码也只能获取被保护的代码和数据的密文。如果操作系统内核或是其他恶意的代码试图修改加密的代码和数据，目标程序将会终止运行。通过双影子页表机制，Cherub 可以检查 PEX 的入口点和退出点来防止跳转攻击。控制权在可信和不可信代码间的转移只能通过预先定义的入口点和退出点。通过度量表单和载入时的完整性检查，Cherub 可以防止恶意代码对程序磁盘镜像进行修改和替换。

Cherub 可以利用设备 I/O 虚拟化[8]或是软件机制[80]来防止恶意代码利用 DMA 设备进行攻击，然而 Cherub 不能解决目标程序的逻辑或是语义漏洞。特别的，如果被保护代码使用不可信代码提供的服务，恶意代码可以通过修改未被保护的代码来改变被保护代码行为的正确性。保护代码访问未被保护的数据同样会带来被攻击的风险，Cherub 的设计不能防止此类威胁。

通用目的的虚拟机监控器所能够提供的安全保证局限于其庞大的 TCB。Cherub 的 LVMM 拥有非常小的 TCB，通过动态可信度量根技术，Cherub 将包括 BIOS、OS 引导模块、DMA 的硬件设备和操作系统内核都排除到 TCB 之外。虽然，Cherub 的

启动模块暴露到了不可信的操作系统之中，恶意代码在 LVMM 启动以前，可以修改启动模块和 LVMM 的镜像文件，Cherub 不能够阻止这种类型的攻击。然而，Cherub 以 LVMM 作为 MLE，其完整性将被动态可信度量根度量并且度量值将保存到 TPM 中。任何对 LVMM 的修改都可以通过查看 TPM 中的 PCR 值检测出来。采用同样的方式，Cherub 可以检查出对度量表单和从启动模块传递的参数的任何更改。

Cherub 可以使用加密机制和通过追踪目标程序的系统调用将文件 I/O 转化成内存访问 I/O 来实现安全文件 I/O 路径。Cherub 可以采用和已有研究相似的方式[74][75]在文件数据被目标程序读取时将其解密，而当其重新写回磁盘上时将其加密。

4.6.2 性能

本文在 Lenovo Thinkpad X200 上测试了 Cherub 原型系统的性能， X200 拥有 Intel 酷 2 双核主频 2.53GHz 的 P8700 处理器，该处理器同时支持 VT 和 TXT 技术。X200 同时拥有 2GB RAM，并集成了 Intel TPM。操作系统采用的 Debian Lenny Linux 内核版本 2.6.28。Cherub 的原型系统只有 4326 行代码，其中 LVMM 仅有 2682 行。本实验测量了 LVMM 加载、运行和保护目标程序的开销。

4.6.2.1 LVMM 启动开销

实验测量了准备 LVMM、调用 GETSEC [SENTER]指令以及初始化 LVMM 的时间开销。测量使用 RDTSC 指令计算 CPU 的周期数，然后依据 CPU 的频率换算成时间。相同的测试进行了 50 次，从载入启动模块到恢复操作系统平均需要 91ms。在同样硬件配置下，还测试了 Xen（ Xen-3.0.1，内核版本 2.6.12.6）的启动时间。表 4.1 列出了主要的测试结果。

表 4.1 LVMM 启动开销

操作	时间
准备 LVMM	57ms
调用GETSEC [SENTER]	16ms
初始化LVMM	18ms
总计	91ms
Xen加电启动	34S

从表 4.1 中可以看出，即使不对 Xen 的启动过程进行静态度量，Xen 的加电启动仍需大约 34s。而 LVMM 的动态加载和度量时间只有 91ms，远远小于通用虚拟机管理器的启动时间。

4.6.2.2 LVMM 运行开销

实验从 SPEC CPU2000 整数运算的基准测试程序中选取了处理器和内存密集型的程序作为本实验的测试用例。通过比较这些程序在不同情况下的运行时间来测试 Cherub 在运行时和保护目标程序时带来的性能开销。如图 4.8 所示，将这些基准测试程序在普通操作系统中的运行时间标准化为 1，同这些程序在 LVMM 加载后但未提供保护功能和提供保护功能两种情况下的运行时间做比较。从图中可以看出这两种情况下 Cherub 对性能的影响。标签“LVMM-no-protect”表示 Cherub 加载后未提供保护功能时，运行这些程序的时间开销。标签 “LVMM-protect” 表示的是目标程序被保护时的时间开销。

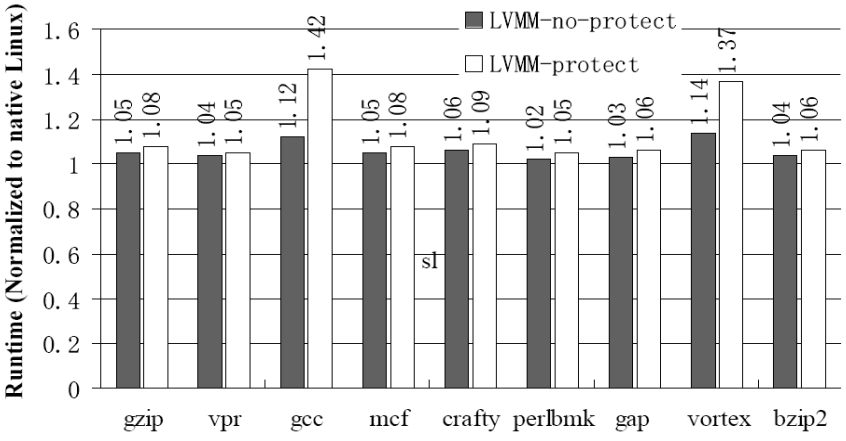

图 4. 8 LVMM 虚拟化和保护功能的基准测试结果

从图 4.8 中可以看出，LVMM 的虚拟化开销是非常小的，所有的基准测试程序只增加了不到 5%的开销。LVMM 对目标程序的保护开销同样比较小，除了 GCC 和 Vortex 外，大多数的基准测试程序增加的开销小于 10%。测试结果表明了 Cherub 轻量级设计的优势和优化技术减小加解密开销的有效性。

LVMM 保护功能的开销主要来自于以下两个方面：1）追踪目标程序的执行过程以验证进出 PEX 的合法性；2）同步加密的页面。为了分析不同的因素对性能的影

响，实验首先取消了 CR3 在 SPTs 和 PSPTs 之间转换时的加解密操作来测试追踪目标程序的开销。这样性能的开销就主要来自于中断、系统调用或是用户态可信代码与不可信代码间相互调用时引起的 CR3 在 SPTs 和 PSPTs 之间的转换开销。

在大多数情况下，目标程序的被保护部分只占整个目标程序很小的一部分。同中断和系统调用引起的 CR3 在 SPTs 和 PSPTs 之间转换相比，用户态的可信代码与不可信代码引起的转换频率很小，因此可以忽略。同时尽管中断会引起 CR3 转换，但是同现有的处理周期相比，中断的频率相当的小，因此中断带来的开销也非常的小，故追踪的开销主要来自于系统调用。

为了评估 Cherub 在保护目标程序时，系统调用带来的开销，本文使用特殊 SPTs 和 PSPts 运行与图 4.8 中相同的基准测试程序。通过设置相关内核页表的可执行位，使中断和用户态的可信代码与不可信代码的转换不再引起 CR3 的切换，图 4.9 给出了测试的结果。如同前文分析的那样，系统调用引起的开销是追踪目标程序开销的主要来源。当加解密的开销非常的小时，系统调用的开销占总开销的近 80%。

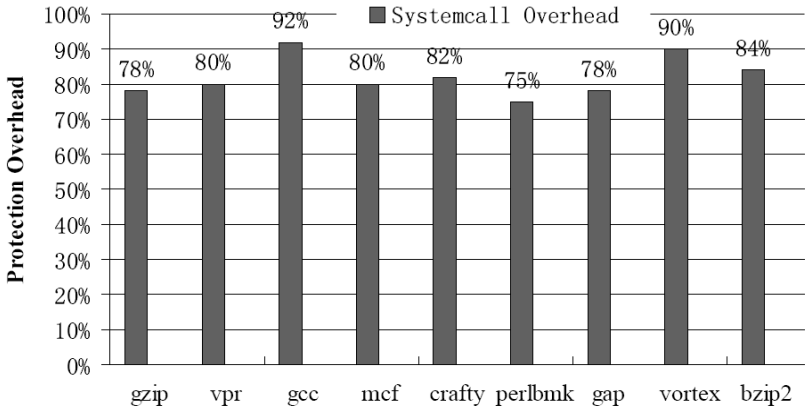

图 4.9 系统调用开销的基准测试结果

当需要同步的加密页面数量非常少时，加解密的开销是非常小的。然而随着需要同步的加解密页面的增多，性能的开销也逐步增大。为了进一步分析性能开销与同步加解密页面间的关系。本文使用了一个可以在运行时改变修改内存大小的测试程序。

图 4.10 给出了该程序的测试结果，运行时的开销随着需要同步的内存数量增加而增大。如果动态的内存修改足够小（本测试中小于 3MB），则同步开销小于 20%。

但是当同步内存达到 8MB 时，同步的开销增加了 200%。因为程序运行时具有局部性，本文认为 Cherub 的保护机制不会带来很大的性能开销，而且上述的结果是在没有优化的情况下得到的。

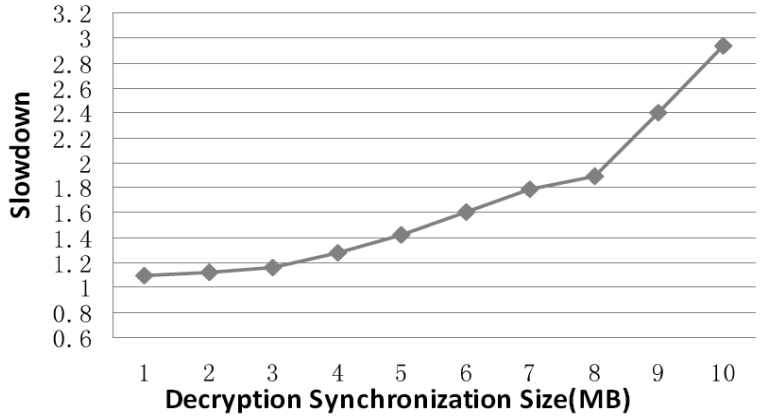

图 4.10 加解密同步开销的基准测试结果

4.7 小结

本章提出了一种基于轻量级虚拟化的动态可信执行环境构建机制—Cherub。Cherub 利用动态可信度量根和硬件虚拟化技术在运行的操作系统下装载一个轻量级的虚拟机监控器。通过该轻量级虚拟机监控器，Cherub 为目标程序创建被保护的执行环境 PEX。Cherub 可以确保被保护代码和数据只有在 PEX 中才能够被执行和访问，并支持用户定义的程序保护策略。Cherub 是一种灵活的设计可以保护整个应用程序或是应用程序中选取的部分代码和数据。实验结果表明了 Cherub 架构的虚拟化及保护功能带来的性能开销都非常小。

5 可信虚拟域间隐形流控制机制

在分布式虚拟机系统中使用可信计算构建联合可信基，并在虚拟机粒度上实施强制访问控制构建可信虚拟域能够提升以虚拟机监控软件做为底层架构系统的安全性，但是目前在可信虚拟域间实施强制访问控制的系统仅能控制虚拟机间公开的信息流，不能防止可信虚拟域间通过隐通道传递信息。第二章给出了能在可信虚拟域间消除隐形流威胁的优先中国墙模型，本章将研究可信虚拟域间的访问控制实施机制。

5.1 研究背景

根据研究公司 IDC 的权威报告[104]，虚拟化技术已经成为当前 IT 基础架构中的核心技术并对市场有着巨大的吸引力。据该公司预测，到 2013 年虚拟机和物理服务器的之比将达到 3：2。虚拟化技术的资源聚合能力可以使多个虚拟机及它们的负载运行于同一个物理节点，但是这同时带来信息泄露的风险。特别是在以虚拟化作为基础架构的云计算环境中，不同企业的虚拟机服务器运行于同一个云环境中，甚至可能有竞争关系的企业虚拟机服务器运行于同一个物理节点之上，在这种应用场景中那怕是数据碎片的泄露也是不能接受的。虚拟机监控软件提供的虚拟机间资源隔离对高安全级的应用是不够的，在分布式虚拟机系统中使用 TCG 的可信启动度量平台软件栈的完整性，并构建跨平台的分布式联合可信基，由联合可信基在虚拟机间实施访问控制是解决这个问题的一个有效方案。将不同节点间的虚拟机组织成不同的可信虚拟域[56]，在不同可信虚拟域间实施不同的安全策略，构建跨平台的可信计算环境。

基于虚拟机架构的系统中有多种构建可信虚拟域的方法，如 NetTop [101]、Shamon [59]、 IBM 的 TVD（Trusted Virtual Domain）[53][54][55]等。可信虚拟域的构建是为了在虚拟域中和虚拟域间实施不同的访问控制策略来满足域内和域间的不同的安全需求。例如有多个公司的虚拟服务器运行于同一云环境下，每个公司的不同部门拥有不同的虚拟服务器。很自然一个公司内部不同部门及公司间的访问控制应该是不同

的。特别是对有着竞争关系的公司之间的服务器，应严格地控制它们之间的信息流防止信息泄露。

在虚拟机系统中，可信虚拟域间的隐通道威胁主要是因为虚拟机间的资源共享。本文第二章分析了现有的访问控制策略不适用于消除可信虚拟域间隐通流的原因，并给出了更加灵活的优先中国墙策略。优先中国墙策略（PCW）是一个基于历史信息的访问控制策略，这需要实施 PCW 的系统能够获取全局和历史信息。本章给出一个支持 PCW 的可信虚拟域间隐形流控制架构 CFC 和 PCW 的实施算法，CFC 能使用 PCW 消除可信虚拟域间的隐形流。

5.2 联合可信基

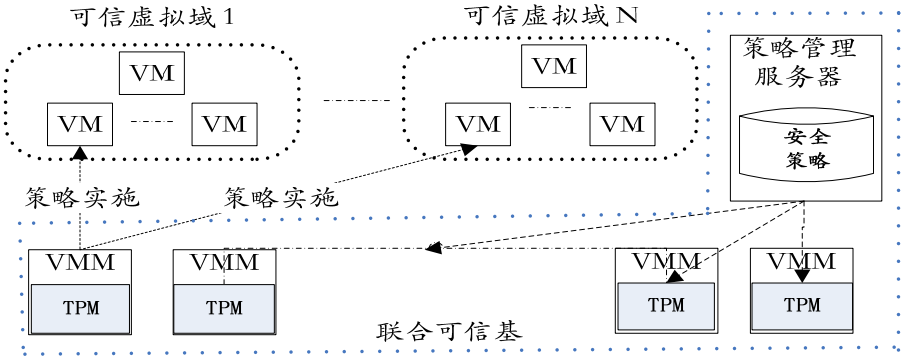

图 5.1　联合可信基

分布式虚拟机系统通过图 5.1 所示的方式建立联合可信基，分布式系统的每个节点通过前文介绍的 TCG 可信启动方式，建立从信任根到虚拟机管理软件的信任链，然后相互验证并扩展信任链于整个系统，形成联合可信基（建立联合可信基的过程采用传统的 TCG 方式，下文将不再累述）。在分布式虚拟机系统中，可信基除了包括每个节点的虚拟机监控器及特权域（如果采用类似 Xen 的虚拟机监控器），还应包括分布式虚拟机系统中的管理节点。本文假定联合可信基自身并不主动地泄露信息，但是联合可信基可能会包含隐通道而被恶意的虚拟机利用来传递违背访问其实施的控制策略。因此，如果一系列虚拟机运行于同一节点之上，它们就可能通过隐通道传递信息。

信息泄露同虚拟机运行时的外部系统环境相关。外部系统环境是指该节点上一系列的虚拟机启动、停止事件，包括虚拟机的载入序列和正在运行的虚拟机。在分布式环境中，还包括虚拟机的迁移和跨节点的虚拟机间通信事件。因此如果要防止隐形流，就需要控制各个节点上的虚拟机启动、停止、迁移和通信。同时本文假设一旦一个 VM 终止运行，TCB 完全消除该虚拟机内的任何信息。这意味着虚拟机必须在运行时才可能通过隐通道传递信息，一旦一个虚拟机停止运行，则不会泄露任何信息。本文还假定，允许可信虚拟域内存在隐形流，而在可信虚拟间控制隐形流，消除不符合安全策略的隐形流。

上述的假设符合可信虚拟域的实际使用需求，例如在采用虚拟化作为基础架构的云计算环境中，云服务的使用者总是假定云服务的提供者是善意的，不会主动去泄露使用者的信息，同时一旦一个用户停止使用该云服务，则该信息不应该继续保留在该云环境中。在云环境中可以将同一公司的不同部门间的虚拟服务器设为一个可信虚拟域，而不同公司的虚拟机服务器隶属于不同的可信虚拟域，因为通常可以容忍同一公司不同部门间存在隐通道的可能，而必须防止不同公司间的任何信息流。

5.3 设计架构与实施算法

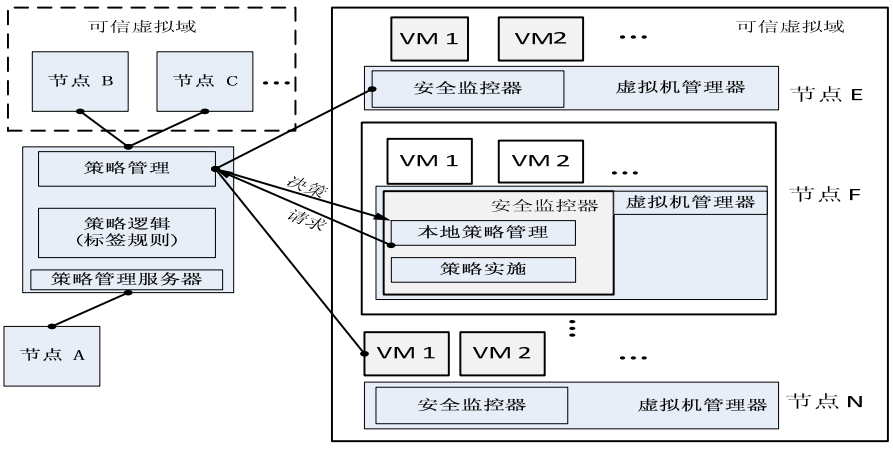

图 5.2　CFC 系统架构

CFC 采用 Flask[102]的设计思想，将访问控制的策略和实现机制进行分离。CFC 的主要设计目标是弥补现有支持强制访问控制的可信虚拟域架构的不足，消除可信

虚拟域间的隐形流问题。依据第二章的分析，虚拟机系统中可能存在两种隐形流，一种是直接隐形流，另一种间接隐形流。CFC 使用第二章中给出优先中国墙策略（PCW）作为本章架构的域间访问控制模型。PCW 是一个基于历史信息的访问控制策略，这要求 CFC 能够记录 VM 的运行历史和获取任意节点的信息。同时 CFC 还应该在虚拟域内支持多种访问控制策略。CFC 的次要设计目标包括保持对虚拟机的透明性，支持多目标的调度策略及较小的性能开销。

图 5.2 描述了 CFC 系统的各个子模块。策略管理服务器（OSS）是一个特殊的节点，用于收集全局信息（包括每个节点的虚拟机启动、迁移和通信信息），并提供策略制定及虚拟机标签分配的用户接口。策略的实施依赖于每个节点的安全监控器，安全监控器（SR）监控本地节点上虚拟机的启动、迁移和通信，并依据全局的虚拟机运行历史，对虚拟机启动及在节点间迁移和虚拟机间通信做出许可判定，并控制上述的虚拟机操作和行为。

5.3.1.策略管理服务器

策略管理服务器主要包括策略管理模块和策略逻辑模块。策略管理模块更新和同步相应节点中缓存的策略信息。策略逻辑模块用于管理全局的历史运行信息。节点安全监控器的请求主要包含两大类：一类是注册请求，另一类则是对节点间虚拟机信息的同步请求。节点安全监控器需要向策略管理模块进行注册来表明该节点希望加入联合可信基并实施策略管理模块的访问控制策略。为了保持策略管理服务器和各个节点上的安全监控器的状态保持一致，当策略改变或是系统的状态发生变化时，策略管理模块将通过同步算法使向其注册的安全监控器更新状态。

5.3.2.安全监控器

安全监控器不仅要保存本地节点的策略信息，还需要对节点内的行为进行策略决策。 安全监控器包括多个模块：本地策略管理模块（Local Policy Management Module，LPMM）存储同本地节点上虚拟机相关的策略逻辑，负责同策略管理服务器交互，当本地策略缓冲模块发生缺失时，该模块向策略管理服务器请求更新信息，并维护二进制形式的策略上下文，对于本地节点上的虚拟机启动、迁移和通信请求

做出判定。安全控制模块（Security Control Module）负责判定的执行。

5.3.3 标签和策略制定

```
<ChineseWall priority="PrimaryPolicyComponent">
    <ChineseWallTypes>     //中国墙策略类型
        <Type>T1</Type> //类型 T1
        <Type>T2</Type> //类型 T2
        <Type>T3</Type> //类型 T3
        <Type>T4</Type> //类型 T4
    </ChineseWallTypes>

    <ConflictSets> //冲突集
        <Conflict name="RER1">    //冲突集名称 RER1
            <Type>T1</Type>
            <Type>T3</Type>
        </Conflict>
        <Conflict name="RER2">    //冲突集名称 RER2
            <Type>T1</Type>
            <Type>T4</Type>
        </Conflict>
```

图 5.3 优先中国墙策略类型和规则定义示例

CFC 允许系统管理者通过一系列的标签和策略描述文件来定义优先中国墙策略。例如有以下中国墙标签{T1,T2,T3,T4}。标签是以虚拟机为分配单元的，因此下文将以虚拟机的标签代表被分派了某个标签的虚拟机。不能存在信息流的虚拟机之间，将提供过第二章中介绍的冲突集（Conflict of Interest Set，CIS）进行描述。如使用上述的标签，系统要确保 T1 与 T3、T4 之间不能存在隐形流，则可以将 T1 与 T3、T1 与 T4 定义为冲突集。需要注意的是依据第二章中对优先中国墙策略的形式化描述，这种冲突关系是对称的，即如果 T1 与 T2 冲突，反之 T2 与 T1 也冲突。但这种冲突关系通常不是自反与传递的，故可以将与 T1 有冲突关系的虚拟机标签的集合表示为 CIS(T1)={T3,T4}，同时 T1 属于 CIS(T3)和 CIS(T4)。

策略管理服务器的策略管理模块提供标签分派和策略制定接口。CFC 采用 XML 文档使用 sHype 的中定义的 XML 元素。图 5.3 给出了上例中的标签和策略描述文件。

同 Flask 的设计思想不同的是 CFC 中访问控制标签和访问控制客体（VM）之间是通过虚拟机配置文件绑定的。这样设计的原因是分布式虚拟机架构是以虚拟机为

控制粒度的，相比操作系统中文件和进程的控制粒度，虚拟机粒度下的客体数量非常的少，将标签与客体绑定可以兼容现有虚拟机监控软件的管理方式，减少同步策略的开销。

5.3.4 实施算法

```
If (HCWTA1为空) {
        允许任意标签的虚拟机启动;
        将该虚拟机的标签加入本地节点的HCWTA1中;
        更新该节点HCWTA1在OSS中的对应项;
 }
 else {
        同步OSS更新本地节点的HCWTA1;
        加锁OSS中该节点HCWTA1的相关项;
  if (VM的标签在HCWTA1中 ){
        允许该VM在本地节点启动;
  }
    else if (VM的标签不与HCWTA1中的标签冲突 ) {
        允许该VM在本地节点启动;
        将该VM的标签加入本地的HCWTA1;
        更新OSS中该节点的相关数据;
          }
     else{
           不允许该VM在本节点启动;
           }
        解锁OSS中该节点HCWTA1的相关项;
 }
```

图 5.4 虚拟机启动算法

本地的策略管理模块依据系统中虚拟机运行的历史信息和系统管理者指定的策略形成控制决策。依据第二章的形式化分析，最终在系统中会形成多个可信虚拟域（动态联盟），在策略逻辑模块中使用历史中国墙数组（HCWTA）来记录属于同一域的虚拟机标签。全部的历史中国墙类型数组组成历史中国墙类型表（History CW Types Table，HCWTT）。HCWTT 只存在于策略服务器中，而在每个节点的策略管理模块中都保存同该节点上运行的虚拟机标签相关的历史中国墙数组。

图 5.4 描述了判定虚拟机启动的算法。假定节点 N1 上有标签为 T1 的虚拟机，该节点上的中国墙数组为 HCWTA1。从该算法中可以看出该节点的策略管理模块需要同 OSS 同步来获取同一个可信虚拟域中所有的标签。这些信息被本地策略管理模

块用于推断是否 T1 同所有处于该节点上虚拟机所属的可信虚拟域中的其他标签冲突。同时可以看到，当本地节点上的虚拟机状态发生改变时。本地的策略管理模块将通知策略管理服务器，更新历史中国墙类型表中的对应信息。为了保持所有处于同一个可信虚拟域中其它节点状态的一致性。策略管理服务器的策略逻辑模块将会锁住 HCWTT 中对应的 HCWTA，直到一个节点完成同步操作。

```
依据OSS中的对应项更新HCWTA1和HCWTA2
加锁OSS中的对应项;
if (HCWTA1 == HCWTA2){
    允许VM的迁移或是通信请求;
}
else{
  if( T ∈HCWTA1,T∈∪{CIS(x)|x∈HCWTA2}) {
    拒绝VM的迁移或是通信请求;
  }
  else {
    允许VM的迁移或是通信请求;
    跟新OSS中HCWTT同HCWTA1和HCWTA2的对应项为HCWTA1∪HCWTA2;
    更新HCWTA1和HCWTA2为HCWTA1∪HCWTA2;
  }
解锁OSS中的对应项;
```

图 5.5 虚拟机迁移和通信算法

图 5.5 描述了虚拟机在节点间迁移和通信的判定算法。算法中假设如下：存在节点 N1 和 N2，它们对应的中国墙数组分别为 HCWTA1 和 HCWTA2。节点 N1 上的虚拟机需要同节点 N2 上虚拟机通信或是 N1 节点的虚拟机需要迁移到 N2 节点上。同虚拟机启动的算法相似，为保持在一个可信虚拟域中各个节点状态的一致性而进行加锁和同步操作。

当一个虚拟机关闭时，该节点的安全控制模块需要检测该虚拟机是否是该节点运行的最后一个虚拟机。因为依据第二章中的优先中国墙策略和图 5.4 及图 5.5 的算法，可以看出一个节点不能同时运行属于不同可信虚拟域的虚拟机。这意味着在某一个时刻，该节点属于某一个可信虚拟域或者没有被任何的可信虚拟域所占据。因此，一旦这个节点上没有虚拟机运行，该节点就从它隶属的可信虚拟域中释放出来，

可以加入其它可信虚拟域。因此虚拟机关闭的算法可以描述如下：判断 VMA 是否为节点 N1 上最后一个正在运行的虚拟机：若 VMA 不是，关闭 VMA，不改变该节点上的 HCWTA；否则判断节点 N1 是否为其所属可信虚拟域中最后一个正在运行虚拟机的节点：若 N1 不是，关闭 VMA，清空本地 HCWTA1；若 N1 是，则关闭 VMA，清空本地 HCWTA1，同步 OSS，清空在 OSS 中 HCWTT 内与 HCWTA1 相对应的 HCWTA。

5.4 主要机制

本文采用支持 Xen 安全模块（XSM）的 Xen 虚拟机监控器构建 CFC 原型系统的本地安全监控器，而策略管理服务器的功能被实现为 Linux 操作系统中的几个相互协作的进程。图 5.6 给出 CFC 基于 Xen 虚拟机监控器的实现架构。

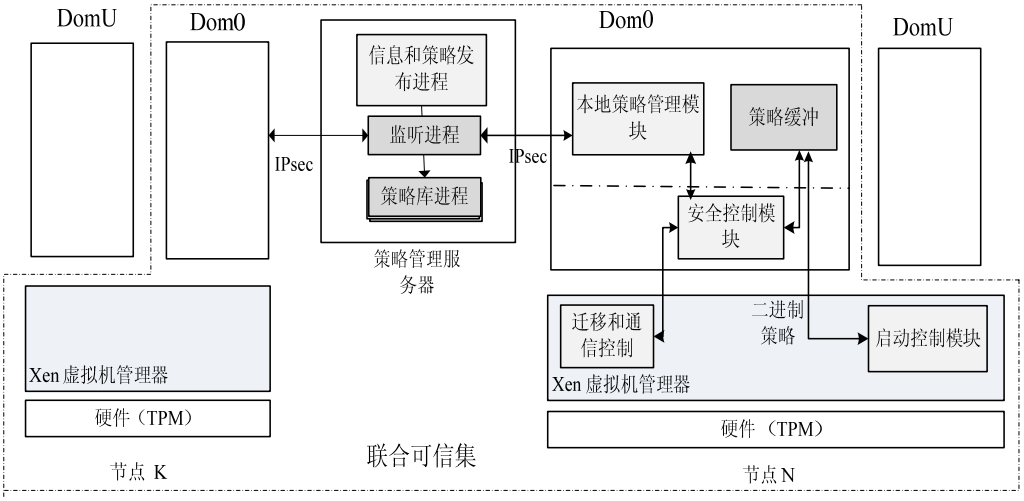

图 5.6 基于 Xen 虚拟机监控器的实现架构

5.4.1 信息收集和策略发布

信息和策略发布进程提供策略管理功能的图形化界面，能够解析管理员以 XML 格式发布的策略，并保存到策略库，也能接收节点返回的策略执行信息并以友好的方式显示给管理员，同时可以将包含标签的虚拟机配置文件发布到指定节点中去。

监听进程作为 Linux 的守护进程，在操作系统启动后加载。监听进程监听指定的端口等待本地节点的安全监控器向策略管理模块注册。当监听程序接收到注册信息后将这些信息交给策略库进程，并将策略库中保存的策略信息发给该节点的本地策略管理模块。策略库进程维护着一个全局的数据结构，记录着向策略管理模块注册的节点中 VM 的运行情况。

策略库包含所有节点中虚拟机所属的可信虚拟域信息，即包含各个节点 HCWTA。策略库通过一个结构体数组来组织所有节点的节点信息并通过节点 IP 进行索引。策略信息库组织结构如图 5.7 所示。其中，节点 0 为一个新注册的节点，该节点尚未运行任何的虚拟机，故其对应的 HCWTA0 为空。多个节点可能指向同一个 HCWTA，例如节点 2 和节点 N 上的虚拟机属于同一个可信虚拟域。

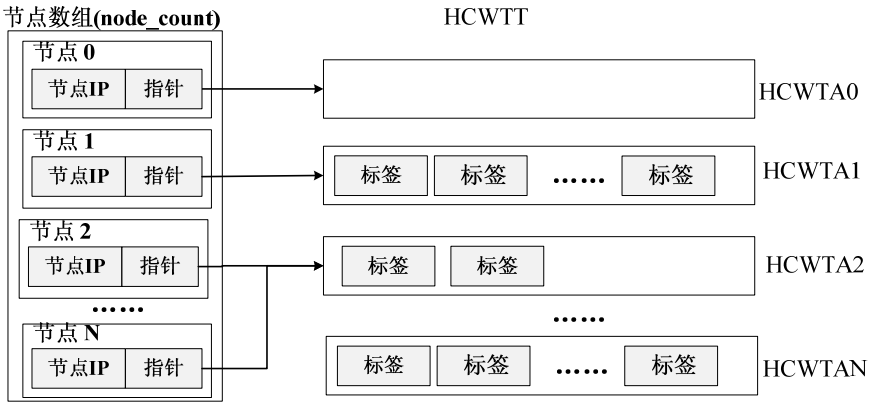

图 5.7 策略库组织结构

5.4.2 本地策略管理

在本地节点的策略管理模块中管理着与本地节点上虚拟机相关的可信虚拟域相关信息。由于节点间策略决策涉及同策略管理服务器的交互，为了保证本地安全监控器能够顺利完成策略实施，本地策略管理模块中还需保存一些与节点相关的信息（如已运行虚拟机列表和节点 IP 等）以及与虚拟机相关的信息（虚拟机 ID、虚拟机 IP 等）。通过这些附加信息，本地策略管理模块能够方便地获得整个系统的全景图，进而对虚拟机的行为进行决策，节点策略信息组织结构如图 5.8 所示。

节点信息结构体中的 PCW 策略信息来自本地策略管理模块，节点 IP 即为当前节点中 Dom0 的 IP，在对节点间虚拟机迁移进行策略决策时需要用到。最大虚拟机数和虚拟机列表记录当前在节点中运行的虚拟机，当虚拟机的运行状态发生改变时，该列表也需要发生相应的改变。虚拟机信息结构体中主要包含虚拟机 IP、DomID、所属节点 IP。节点策略管理模块中的策略信息是进行节点策略决策的基础，需要时刻保持其信息的准确性和实效性。一旦本地的 PCW 策略信息发生改变，则要第一时间通知并更新节点策略管理服务器的策略信息，反之在完成全局的状态信息发生变化后， 相应的全局信息也应该及时地更新到本地策略管理模块中，在前文实施算法中介绍的加锁机制可以避免因信息不同步造成决策失误。

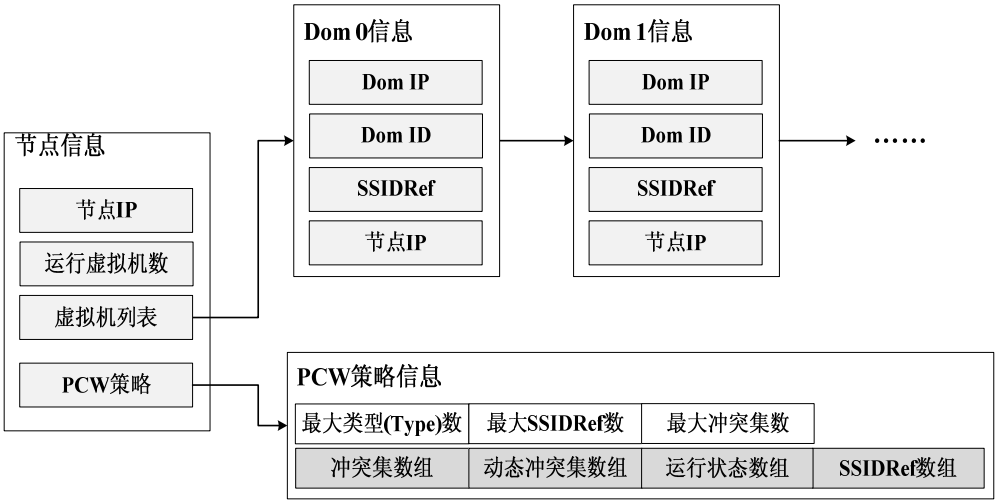

图 5.8 节点策略信息的组织结构

5.4.3 本地策略实施

CFC 的原型系统通过迁移和通信控制及启动控制模块在节点实施 PCW 策略，这两个模块利用 Xen 安全模块（XSM）并在虚拟机的启动及迁移和通信的控制路径上加入钩子函数来实施 PCW 的策略。

在 Xen3.1.0 中，Xen 的访问控制模块被称之为 sHype/ACM（Access Control Module）模块，实现了两种策略机制：中国墙策略（Chinese Wall，CW）和简单类型策略（Simple Type Enforcement，STE）。其中 CW 策略用于节点内虚拟机之间的信息流控制，STE 策略则用于资源控制。Dom0 可以根据具体的需求定制安全策略文

档，并交由 ACM 执行，从而达到对虚拟机的资源进行控制以及对虚拟机之间的信息流进行控制的目的。在 Xen 3.2 之后 Xen 引入了 Xen 安全模块（Xen Security Modules，XSM），在原有 sHype/ACM 模块的基础上增加了 Flask 模块，即采用类似 SELinux 的良好粒度和灵活的强制访问控制方法[103]。

CFC 原型系统使用了 XSM 模块中 CW 策略的部分数据结构，并修改了一些 HOOK 函数来实现 PCW 策略。迁移通信和启动控制使用类似的方法，对于虚拟机启动、迁移和通信需要增加前文描述的与策略服务器的通信操作。下文以启动控制为例说明实施 PCW 策略的实现过程。

XSM 模块使用以下几个数据结构来实施 CW 策略：max_types 指的是中国墙类型的总数，max_ssidrefs 指的是虚拟机域标签 ssid 的总数，max_conflictsets 指的是中国墙类型冲突集的总数。ssidrefs 是一个二维数组，存放着域标签和其对应的中国墙类型的详细类型。conflict_sets 也是一个二维数组，存放的是冲突集中发生冲突的中国墙类型。以上数据结构在装载策略后就是定值，并不会随着运行状态而改变。running_types 是一个一维数组，存放的是当前已运行的虚拟机的中国墙类型，也就是说存放的是当前正在运行中的中国墙类型。conflict_aggregate_set 同样是一个一维数组，存放的是与当前运行中的中国墙类型发生冲突的中国墙类型。中国墙策略实现的核心就是根据虚拟机的 ssidref 和 conflict_aggregate_set 两个数组来进行判断。

PCW 使用和 CW 相同的冲突集定义，然而其访问规则不同。PCW 需要记录虚拟机的访问历史记录。判定虚拟机能否创建、迁移和通信是通过获取虚拟机的 ssidref，再结合 conflict_aggregate_set 数组来确定当前运行的虚拟机是否与该虚拟机发生冲突，这需要在 conflict_aggregate_set 数组里反映出该节点所属可信域中虚拟机运行的历史记录。因此，CFC 原型系统需要修改 chwall_domain_destroy 函数，更新 conflict_aggregate_set 数组的操作，和创建虚拟机的_chwall_pre_domain_create 函数。

对_chwall_pre_domain_create 函数的修改比较复杂，图 5.9 给出了算法思想。首先根据 conflict_aggregate_set 数组来判断是否与当前虚拟机发生冲突，需要修改更新 conflict_aggregate_set 数组数据。不但要将 conflict_aggregate_set 数组里与该虚拟机的所拥有的中国墙类型发生冲突的类型加 1，还要将 conflict_aggregate_set 数组里所有

非 0 数据加 1（也就是说将与当前运行的虚拟机发生冲突的类型的总数加 1）。然后根据虚拟机的 ssidref 和 conflict_sets 来查找与虚拟机所拥有的类型相冲突的类型，同时记录这些类型，再将 conflict_aggregate_set 数组里非 0 数据而且是不在记录的类型里的数据加 1，并将相加后的结果赋值给 m，最后将被记录的类型数据设置成 m，因为这时当前运行的虚拟机也与要创建的虚拟机的相冲突类型相冲突，这样就将虚拟机访问历史记录反映到 conflict_aggregate_set 数组中，至此，更新 conflict_aggregate_set 数组，完成优先中国墙策略创建虚拟机策略实施判定。

```
m=1;
for(i=0;i<冲突集总数;i++){
        common=0;
        for(j=0;j<类型总数;j++)
                寻找在冲突集中与该虚拟机相同的类型;                ;
        }
        if(common==0) continue;
        for(j=0;j<类型总数;j++)
                寻找下一个冲突集;
        }
        if(m==1){
                m=0;
                求出当前运行的类型总数;
    }
    For(j=0;j<类型总数;j++){
        查找该虚拟机所拥有类型与冲突集数组中相冲突的类型;
        if(flag==0){
        m=0;
        更新conflict_aggregate_set数组;
        }
    }
```

图 5.9 优先中国墙策略创建虚拟机时更新 conflict_aggregate_set 的算法

5.5 有效性和性能评测

可信虚拟域间的隐形流控制机制通过实施 PCW 策略控制基于虚拟机架构的分布式环境中一系列的虚拟机启动、停止、迁移和通信事件来消除因为资源共享产生的隐形流。实施访问控制会产生时间开销，其主要包含策略决策时间和策略信息在

节点间的更新时间两个部分。本文在 CFC 的原型系统上进行了如下的实验来分析 CFC 设计和 PCW 实施算法的有效性和开销。

实验环境由五个节点构成，其中策略管理服务器节点拥有 Intel Xeon2 路 4 核 CPU，单核的主频为 1.60GHz，每一个核的高速缓存容量为 4MB，硬盘容量 80GB，物理内存 4GB，其它节点拥有主频为 2.33GHz 的 Intel 双核处理器、2MB 二级缓存、2GB 内存和 80GB 7200RPM 的硬盘，网络出口带宽为 100Mbps。

5.5.1 有效性

假设 CFC 系统上共运行四个不同公司（Oil-A, Oil-B, Bank-C 和 Other）的虚拟机服务器，分别给不同公司的虚拟机以它们公司的名称分派如下的标签 Oil-A, Oil-B, Bank-C 和 Other。其中 Oil-A 和 Oil-B 公司具有竞争关系，所以冲突集定义为{(Oil-A, Oil-B)}，这意味着必须控制拥有 Oil-A 和 拥有 Oil-B 标签的虚拟机之间的任何信息流。在初始阶段，假设每个节点恰好运行着来自于不同公司的虚拟机。

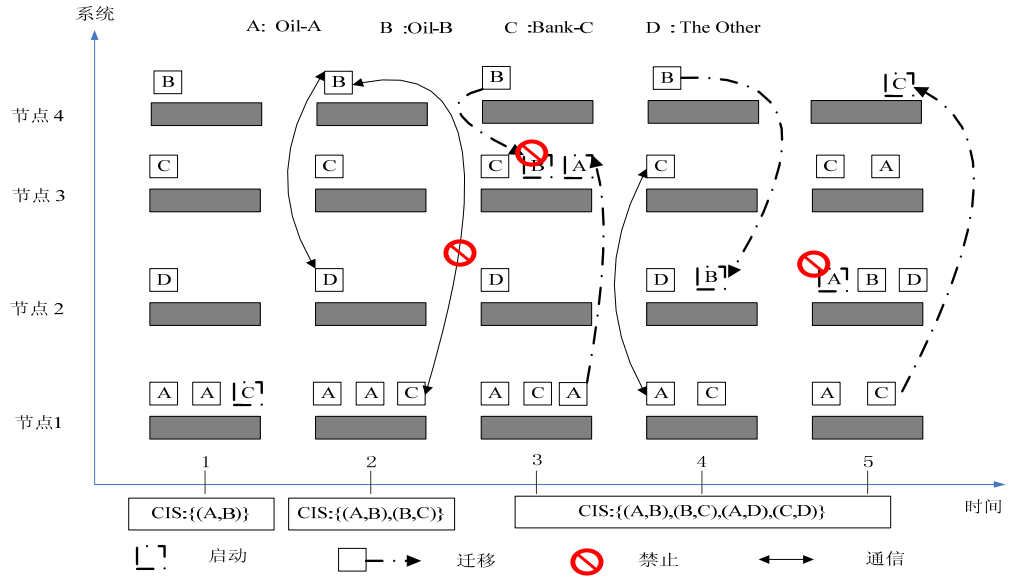

图 5.10 CFC 隐形流控制效果示例

如图 5.10 所示，在时刻 1， 当 Back-C 需要在节点 1 上启动时，此刻的冲突集为 CIS = {(Oil-A, Oil-B)}。所以允许 Bank-C 和 Oil-A 之间存在隐形流，故 Bank-C 被允许在节点 1 上启动。 Bank-C 和 Oil-A 组成了一个可信虚拟域，依照 PCW 策

略它们将拥有相同的利益冲突关系，此刻冲突集变为 CIS ={(Oil-A, Oil-B), (Oil-B, Bank-C)}。在时刻 2，当 Oil-B 试图同 Bank-C 和 Other 通信时，因为 Bank-C 和 Oil-A 处于同一个可信虚拟域中，故 Bank-C 和 Oil-B 利益冲突， Oil-B 和 Bank-C 之间的通信请求被禁止。但是 Oil-B 和 Other 之间没有利益冲突关系，因此它们之间的通信将被允许，Oil-B 和 Other 之间建立了一个可信虚拟域。最终将有两个可信虚拟域 {Oil-A, Bank-C} 和 {Oil-B, Other}。可信虚拟域的建立根据事先定义的冲突关系，由虚拟机的启动、通信或是迁移序列所决定。在时刻 3 和时刻 4，在运行同一个可信虚拟域内节点上的虚拟机迁移和通信是允许的，而不同的可信虚拟域间是禁止的。在时刻 5，节点 4 被释放，可以加入任何一个可信虚拟域，此刻允许 Bank-C 迁移到该节点。

从图 5.10 中可以看出，CFC 通过实施 PCW 策略来调度资源避免安全策略中禁止信息流的虚拟机间的资源共享，从而有效地控制虚拟机启动、迁移和通信过程中因为资源共享产生的直接和间接隐形流。

5.5.2 性能评测

决策及执行优先中国墙策略（策略实施）和同步节点间的策略信息是可信虚拟域间访问控制主要的操作，原型系统的时间开销也主要来源于这两个方面。因此，性能测试将使用这两部分的时间开销来评估可信虚拟域间访问控制对系统总体性能的影响。策略实施时间包括从策略决策模块得出决策结果到执行该决策结果的时间。实验将对 CFC 系统的策略实施时间与 sHype 系统的策略实施时间及进行分析和比较。节点间的更新和同步时间是指在 CFC 系统中，当本地策略信息或策略服务器中的策略信息发生改变时，完成策略信息的更新和同步需要的时间。由于目前没有与之相类似的系统，无法对策略的更新时间做横向对比，只能够进行纵向对比和分析。

在系统中，测量策略实施时间需要从策略决策模块得出决策结果到执行该决策结果为止。为了能够更好的对系统的策略决策开销进行分析，以虚拟机的启动过程为例，分别测试了在原始 Xen 虚拟机系统、sHype 强制访问控制系统和 CFC 系统中

虚拟机启动消耗的 CPU 时钟周期。为了避免多核 CPU 对记录时钟周期造成影响，所有的节点均采用单核运行。

如图 5.11 所示，同没有任何访问控制及 BN 中国墙策略的 sHype 系统相比较，CFC 原型系统在单节点的虚拟机启动时间大约增加了 15%的开销。在 CFC 系统中的 PCW 策略与 sHype 系统中的 BN 中国墙策略不同，不只简单地对静态冲突集数组执行遍历和比对，而且还需要将虚拟机运行序列和静态冲突集数组结合起来形成不断更新的动态冲突集数组(例如 conflict_aggregate_set 等数据结构)。CFC 系统还需要在每次策略决策完成后通过事件通道通知用户空间的安全控制模块将 Xen VMM 内新的策略信息存入策略管理模块的节点间策略信息中，以便进而同步到策略管理服务器中。决策模块需要利用 Xen 提供的事件通道机制完成安全控制模块的策略信息传递工作。Xen 中的事件通道机制类似于操作系统中的异步中断机制，因此策略信息传递消耗的时间也被算在策略执行时间之内。考虑到虚拟机的启动时间和运行时间相比，所占的比例非常小，因此本文认为增加 15%的启动开销是可以接受的。

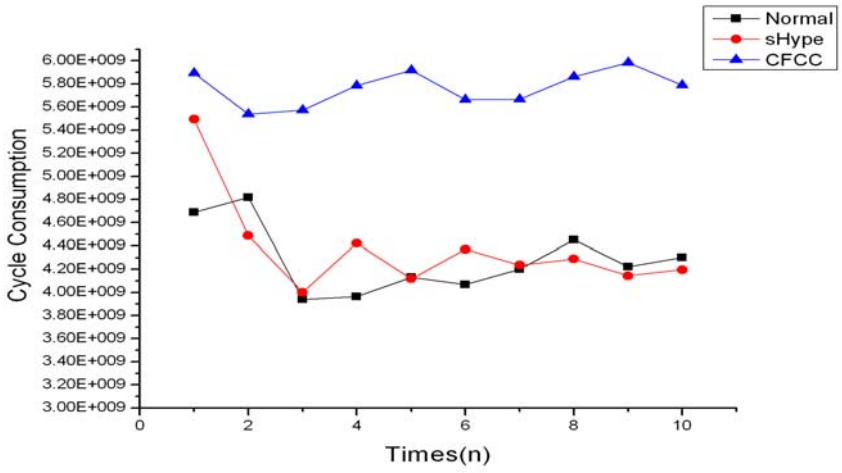

图 5.11 节点虚拟机启动判定开销

运行时的系统对节点间虚拟机的通信，迁移等行为实施强制访问控制。节点间策略决策在形成动态访问域以前须同策略服务器进行同步。与此同时，由于节点策略决策模块进行策略决策而对节点策略信息造成的改变也需要同步各个节点的本地

策略信息。当本地节点的虚拟机启动、节点间虚拟机通信或是迁移时，本地安全监控器同策略服务器之间可能需要进行信息交换。为了避免因信息不同步造成决策失误，信息交换时需要对策略服务器 HCWTT 中相关的数据项进行加锁操作。如果此时其他节点的安全监控器需要访问加锁的数据项，则必须等待该项被解锁。

因此节点间策略信息的同步时间也是影响 CFC 系统性能的主要标志。介于目前的相关系统对分布式环境下虚拟机的行为进行访问控制没有采用中央节点的架构，节点间策略信息的同步时间无法和其它系统进行比较和分析。测试仅以在 CFC 系统的本地节点内执行策略决策的时间消耗为基础，将策略决策时间和策略信息同步时间同未采用 CFC 的系统进行比较。从图 5.12 中可以看出系统的性能随着并发操作的增加而同步降低。幸运的是，在每个节点本地安全监控器中缓存有全局的系统状态，在同一个可信虚拟域内的迁移和通信时不需要同步操作的。因此加锁操作只在系统运行初期建立可信虚拟域时比较频繁，一旦系统运行稳定可信虚拟域形成后，则加锁的同步操作就不再需要。可信虚拟域建立阶段占系统总运行时间的比例同样非常小，因此这种性能的开销也是可以接受的。

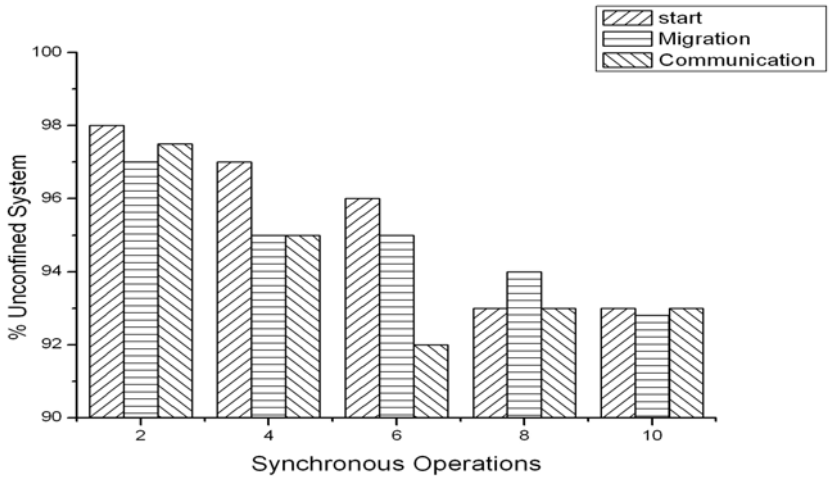

图 5.12 本地安全监控器与策略服务器间的同步开销

5.6 小结

虚拟机之间的共享资源可能带来信息泄露，当虚拟化逐渐成为数据中心、云计算等服务端应用的底层架构时，这种威胁已经成为迫切要解决的安全问题。在虚拟机粒度上实施强制访问控制能够提升以虚拟化做为底层架构系统的安全性，但是目前虚拟机强制访问控制系统还仅能控制虚拟机间公开的信息流，不能防止虚拟机间隐通道的产生，本章结合第二章给出优先中国墙访问控制模型，给出了一种虚拟机环境中可以防止隐形流的实施算法和架构--CFC。CFC 能够满足企业级的安全需要，即依据用户定制的安全策略，确保敏感的信息不会泄露给竞争对手。实验结果表明 CFC 机制对隐形流控制的有效性，而其带来的性能开销是可以接受的。

6 总结与展望

可信计算可以弥补仅依靠软件安全防护方式的不足，从而更好地解决计算机系统安全面临的挑战和问题。然而基于传统计算环境构建的可信环境很难为上层应用提供足够的安全保护，这阻碍了可信计算的应用范围和规模。针对这个问题，本文主要研究在虚拟机架构下如何构建可信计算环境。本文对构建可信计算环境的基本支撑理论进行了研究，给出了可信计算环境的信任链模型及可信虚拟域间隔离模型，并对虚拟机架构下透明信任链构建、完整性保护、程序可信执行环境和可信域间隐形流控制等提出了有效的解决方案，本文的主要创新包括以下几点：

1）本文给出了基于实体依赖关系的可信计算环境的信任链模型。该模型给出了可信状态、信任根、信任度量和信任链的形式化定义，并证明了满足该信任链定义系统的可信性。该信任链模型具有普遍意义，可以评估基于静态可信度量根和动态可信度量根构建的信任链的正确性，并为构建基于虚拟机架构的可信计算环境提供了理论上的支撑。

2）在基于实体依赖关系的信任链理论支撑下，本文给出了一种虚拟机架构下的透明信任链机制--TCT，TCT 可以将 TCG 模式的信任链扩展到应用层的同时保持对操作系统透明，TCT 还可以将用户的数据同其期望的计算环境完整性相绑定。TCT 将操作系统排除在可信基外，解决了可信环境构建中信任链过长的问题，使现有的商用操作系统可以不加任何修改的就具有 TCG 模式的可信启动功能。而且 TCT 可以使一个普通的商业平台转化成只有运行于类似 IBM4758 安全协处理器上的系统才能具有的一些安全特性，例如保护其上计算的机密性。TCT 具有的这些特点可以使其应用于多授权的软件架构中，例如可以用于云计算环境中，保护上层用户的数据隐私和系统的完整性。

3）本文给出了基于轻量级虚拟化的动态可信执行环境构建机制－Cherub。Cherub 是使用动态可信度量根技术的首个虚拟化实现，构建的动态信任链符合可信计算环境的信任链模型中的信任链定义。Cherub 能在运行的操作系统下加载一个可信的轻量级虚拟机监控器（LVMM），由 LVMM 为目标程序创建被保护的执行环境

（PEX），并监控对位于 PEX 中的目标程序被保护页面的访问，从而确保在 PEX 中代码和数据的安全。Cherub 允许由目标程序开发者或是发行商提供保护策略来决定目标程序的保护部分。作为平台的可信基，Cherub 具有很小的代码量并解决了传统虚拟机架构不能动态引导和虚拟化开销大的问题。同时 Cherub 不需要修改客户操作系统和现有应用程序的编程模型，因此能够保护大范围的遗留程序。

4）本文针对可信虚拟域间隐形流控制的需求，提出可信虚拟域间隔离的优先中国墙模型，优先中国墙模型具有不同于 BN 中国墙策略的访问规则，可以根据主体的访问需求建立联盟关系实现冲突关系的动态扩展，从而将客体划分成不同的访问区域，允许敏感信息在不同的域内流动，但不会泄漏到具有冲突关系的客体中。本文给出了优先中国墙模型的形式化描述并证明了该模型能满足 BN 中国墙模型的安全目标。

优先中国墙模型既具有类似中国墙策略在控制公开流时可以依据主体访问需要进行选择的灵活性，又有类似格策略能够将客体分成不同访问集的特点。优先中国墙策略既适用于解决可信虚拟域间的隐形流问题，又可以用于类似 ALDC 等使用中国墙模型创建分布式联盟的系统中来建立更加灵活的访问控制区域。

5）本文给出可信虚拟域间隐形流控制机制--CFC，CFC 能够弥补现有可信虚拟域系统在隐形流控制方面的不足，通过支持优先中国墙策略（PCW）动态构建可信虚拟域并消除虚拟域间的隐形流问题。 CFC 能用于云计算等分布式虚拟机架构中满足企业的高安全需要，依据用户定制的安全策略，确保敏感的信息不会泄露给竞争对手。

最后，本文认为虚拟机架构下的可信计算环境构建还有一些问题需要深入研究：

1）可信度量问题。作为可信计算环境构建的支撑理论，在可信计算语境下，关于信任度量与信任传递的关系还存在诸多的理论问题。例如在信任链构建中应当度量的是实体的行为可信性，而现有的研究包括本文都是在实体的真实性能够无损度量实体的行为可信性前提下得出的。由于目前实体行为的可信性尚不易直接度量，真实性是可信计算实践中的现实方法。采用真实性度量的方式可信性在信任传递过程中是有损失的。问题在于如何衡量这种度量的信任损失以及是否存在无损的信任

度量模型，还是只存在能够满足某些期望的无损信任度量模型，在这方面的研究几乎还是一片空白。

2）可信执行环境调用不可信服务的问题。所有在操作系统内部隔离进程执行空间的安全设计[75][76]，包括本文的 Cherub 机制都面临一个共同的问题--隔离进程执行空间虽然可以保证目标程序的秘密不被恶意的代码获取（包括操作系统内核），却无法保证程序执行的正确性。因为几乎所有的应用程序都需要调用操作系统提供的服务。要解决这个问题，一种可行的方案是由虚拟机监控软件代替操作系统来提供这些的服务，这种方案类似于将资源分割成多个运行独立操作系统的虚拟域而由某一个或是某一些虚拟域来为高安全的应用服务。然而，这种方案会导致虚拟机监控器的代码量因为提供这些服务而膨胀。妥协的方案可能是像 NGSCB 那样，不再支持遗留程序的保护，而是只为少数有高安全需求的应用重新开发接口。这种方案是未来的研究方向，问题是如何筛选应用，如何来设计这种调用接口及提供哪些必备的服务。

3）多目标可信虚拟域中的强制访问控制机制研究。在类似云计算的分布式虚拟机环境中构建可信虚拟域，信息流约束不再是可信虚拟域的唯一目标，还包括诸如性能、吞吐量、节能等多个目标。在诸多的目标下如何进行调度，如何进行综合决策，如何设计可信虚拟域的架构和实施算法也是值得深入探讨的问题。

由于本文属于前沿技术研究，相关的参考文献资料还比较少，所以希望各位专家学者多提宝贵意见和建议，以便开拓和提高今后的研究思路，使本文的研究工作更上一层楼。

致 谢

首先，非常感激我的导师金海教授，论文的完成得益于他的悉心指导和无倦教诲。他以自己敏锐的洞察力、深厚的学术素养、执著的工作热情引导着实验室的研究道路与发展方向，为实验室营造了一个崇尚创新、充满活力的研究氛围。金老师指导我进入科学研究的殿堂，他严谨的科学精神、认真的治学态度、踏实的工作作风、自由的学术风格深深地影响着我，使我受益匪浅，师从这样一位学者让我深感荣幸。在论文即将结束之时，特向金海教授致以诚挚的敬意和衷心的感谢。

真诚感谢邹德清、赵峰副教授和羌卫中博士对我的指导！他们扎实的理论基础、宽广的知识面、敏锐的学术洞察力和精益求精的研究态度对我具有深刻的影响。特别感谢邹老师，他一丝不苟的治学态度，专业的学术造诣，严于律己的一贯作风，诲人不倦的精神让我十分钦佩，受益良多。同时，他经常同我探讨问题，指点和帮助我修改文章，使我顺利地融入到项目组，很快地进入了研究角色。感谢实验室的韩宗芬教授，韩老师热情诚恳，对我们严师慈母般的鼓励和关怀，同时也对我们严格要求，悉心指导，处处为我们着想，深得我们爱戴。感谢实验室的章勤、李胜利、吴松和廖小飞教授对我毕业论文的诸多指导。

真诚感谢三星美国公司的张新文博士，长期的国外求学和研究经历使他具有广博的知识和深厚的学术造诣。感谢他倾囊相授做研究的方法。他认真仔细帮我修改文章，一次又一次地和我探讨文章中的疑点，这些都使我非常感动。

感谢来自贝宁的博士生 Alex，他对人生的信念和乐观精神让我钦佩不已。感谢他把我当作兄弟般的知己，总是海阔天空、无所不谈。

感谢曾经和我在一起工作和学习过的虚拟化安全组的同学们，他们是：项国富、代炜琦、陈刚、杨卫平、龙锦就、石磊、蒋雅丽、李敏、杨凯、王圣兰、詹金波、段培、郑伟德、胡刚、秦攀等，他们曾和我朝夕相处，讨论问题，并给我带来无限快乐。感谢同级的博士生钟阿林、邓莉、刘超、胡亚斌、刘海坤、王晓静、窦亚玲，求学路上，有他们做伴，给了我很多鼓励和帮助。

最后，我要深深感激我的家人。感激我的父母，他们把人世间最无私、最伟大的爱全给了我。自从我考上大学，十几年来一直没有在他们身边，没有尽到为人子的责任，而他们给我了极大的包容，时刻为我的生活操心。在我考上博士的那年，我的父亲却离开了我，但他永远活在我的心里，每当我人生感到迷茫的时候，我都会回想他曾经的教诲，总能从中寻求到答案，坚定人生的方向。

感谢我的岳父岳母，他们无微不至的关怀与照顾，令我终身难忘！

感谢我的妻子，她承担很多本来应该由我承担的责任。衷心感谢她对我的理解和支持。几年来，特别是当我感到疲惫而想与困难妥协的时候，她给我的鼓励和爱，使我有了坚持下去的勇气。感谢我的孩子，他为我的人生平添了太多的快乐，使我第一次感到承担责任的幸福。

感谢所有关心帮助过我的亲人、朋友、老师和同学，衷心地祝愿他们永远幸福安康！

最后，对在百忙之中对我的论文进行评审并提出宝贵意见的各位专家、教授致以诚挚的谢意。

参考文献

[1] Trusted Computing Group. TCG 1_4 Architecture Overview. Available: http://www.trustedcomputinggroup.org/files/resource_files/AC652DE1-1D09-3519-ADA026A0C05CFAC2/TCG_1_4_Architecture_Overview.pdf.

[2] ISO. ISO/IEC15408. Available: http://www.iso.org/iso/iso_catalogue/ catalogue_ics/catalogue_detail_ics.htm?csnumber=50341.

[3] 屈延文. 软件行为学. 北京: 电子工业出版社, 2004, 7-9.

[4] RFC3280&RFC2510. Internet x.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile. 1999. Available: http://www.ietf.org/rfc/rfc3280.txt.

[5] 张焕国, 罗 捷, 金 刚等. 可信计算研究进展. 武汉大学学报(理学版). 2006, 52(5) : 513-518.

[6] 闵应骅. 请不要将信任计算混同于可信计算.中国计算机学会通讯. 2009, 5(12): 78-80.

[7] Leung F, Neiger G, Rodgers D, et al. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization. Intel Technology Journal, 2006. Available: http://www.intel.com/technology/itj/2006/v10i3/

[8] Abramson D, Jackson J, Muthrasanallur S, et al. Intel Virtualization Technology for Directed I/O. Intel Technology Journal, 2006. Available: http://www.intel.com/technology/itj/2006/v10i3/.

[9] Advanced Micro Devices. AMD64 Architecture Programmer's Manual Volume 2: System Programming rev.3.14. 2007.

[10] 金海等. 计算系统虚拟化--原理与应用. 北京: 清华大学出版社, 2008, 4-16.

[11] Rosenblum M. VMware’s Virtual Platform: A Virtual Machine Monitor for Commodity PCs. In: Proceedings of Hot Chips 11. Stanford, CA, USA, 1999, 185-196.

[12] Microsoft. Microsoft virtual pc. Available: http://www.microsoft.com/windows/virtualpc / default.mspx. 2004.

[13] Whitaker A, Shaw M, Gribble S. Scale and Performance in the Denali Isolation Kernel. In: Proceedings of the 5th Symposium on Operating System Design and Implementation. Boston, MA, USA, 2002, 195-205.

[14] Dike J. A User-Mode Port of the Linux Kernel. In: Proceedings of the USENIX Annual Linux Showcase and Conference. Atlanta, GA, USA, 2000, 7-7.

[15] Riehle D, Fraleigh S, Bucka-Lassen D, et al. The Architecture of a UML Virtual Machine. In: Proceedings of the 2001 Conference on Object-Oriented Programming Systems, Languages, and Applications. Tampa Bay, FL, USA, 2001, 327-341.

[16] Kivity A, Kamay Y, Laor D, et al. KVM: the Linux Virtual Machine Monitor. In: Proceedings of the Linux Symposium. Ottawa, Ontario, Canada, 2007, 225-230.

[17] Barham P, Dragovic B, Fraser K, et al. Xen and the Art of Virtualization. In: Proceedings of the 19th Symposium on Operating Systems Principles. Bolton Landing, NY, USA, 2003, 164-177.

[18] Trusted Computing Group. Trusted Platform Module Main Specification, Part 1: Design principles, Part 2: TPM structures, Part 3: Commands. Version 1.2.

[19] Blaze M, Feigenbaum J, Lacy J. Decentralized Trust Management. In: Proceedings of the 1996 IEEE Symposium on Security and Privacy. 1996, 164-173.

[20] Jøsang A. A Logic for Uncertain Probabilities. International Journal of Uncertainty Fuzziness and Knowledge Based Systems, 2001, 9(3): 279-312.

[21] Jøsang A, Knapskog S J. A Metric for Trusted Systems. In: Proceedings of the 21st National Security Conference. NSA, 1998, 16-29.

[22] Jøsang A. Trust-Based Decision Making for Electronic Transaction. In: Proceedings of the Fourth Nordic Workshop on Secure Computer Systems. 1999.

[23] Beth T, Borcherding M, Klein B. Valuation of Trust in Open Networks. In: Proceedings of the 1994 European Symposium on Research in Computer Security. 1994, 3-18.

[24] Meyer J F. On Evaluating the Performability of Degradable Computing Systems. IEEE Transactions on Computers, 1980, C-29(8):720-731.

[25] Isermann R. Process Fault Detection Based on Modeling and Estimation Methods-a Survey. Automatica, 1984, Volume 20: 387-404.

[26] Arlat J, Costes A, Crouzet Y, et al. Fault Injection and Dependability Evaluation of Fault-Tolerant Systems. IEEE Transactions on Computers, 1993, 42(8): 913-923.

[27] Smith S W. Outbound Authentication for Programmable Secure Coprocessors. In: Proceedings of the 7th European Symposium on Research in Computer Security. London, UK, 2002, 72-89.

[28] Abadi M, Wobber T. A Logical Account of NGSCB. Lecture Notes in Computer Science, 2004, Volume 3235: 1-12.

[29] Chen S, Wen Y, Zhao H. Formal Analysis of Secure Bootstrap in Trusted Computing. Lecture Notes in Computer Science, 2007, Volume 4610: 352-360.

[30] Qu W, Li M, Weng C. An Active Trusted Model for Virtual Machine Systems. In: 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications. Chengdu, China, 2009, 145 -152.

[31] Dyer J G, Lindemann M, Perez R, et al. Building the IBM 4758 Secure Coprocessor. IEEE Computer, 2001, 34(10):57-66.

[32] Lie D, Thekkath C, Mitchell M, et al. Architectureal Support for Copy and Tamper Resistant Software. In: Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems. Cambridge, MA, USA, 2000, 168-177.

[33] Suh G, Clarke D, Gassend B, et al. AEGIS: Architecture for Tamper-Evident and Tamper Resistant Processing. In: Proceedings of the 17th International Conference on Supercomputing. San Francisco, CA, USA, 2003, 160-171.

[34] Chen B, Morris T. Certifying Program Execution with Secure Processors. In: 9th Hot Topics in Operating Systems. Lihue, Hawaii, USA, 2003.

[35] Intel Corporation. Intel Trusted Execution Technology Software Development Guide. Available: http://www.intel.com/technology/security/downloads/315168.htm.

[36] Advanced Micro Devices. AMD64 Virtualization: Secure Virtual Machine Architecture Reference Manual. AMD Publication no. 33047, rev. 3.01, May 2005.

[37] Marchesini J, Smith S. SHEMP: Secure Hardware Enhanced MyProxy. In: Proceedings of Third Annual Conference on Privacy, Security and Trust. New Brunswick, Canada, 2005.

[38] Trustedgrub. Available: http://sourceforge.net/projects/trustedgrub.

[39] MacDonald R, Smith S, Marchesini J, et al. Bear: An Open-Source Virtual Secure Coprocessor Based on TCPA. Technical Report TR2003-471, Department of Computer Science/Dartmouth PKI Lab Dartmouth College, 2003.

[40] Marchesini J, Smith S, Wild O, et al. Experimenting with TCPA/TCG Hardware, or: How I Learned to Stop Worrying and Love the Bear. Technical Report TR2003-476, Department of Computer Science, Dartmouth College, Hanover, New Hampshire, December 2003.

[41] Sailer R, Zhang X, Jaeger T, et al. Design and Implementation of a TCG-based Integrity Measurement Architecture. In: Proceedings of the 13th Conference on USENIX Security Symposium. Berkeley, CA, USA, 2004, 223-238.

[42] Open Secure Loader. Available: http://os.inf.tu-dresden.de/~kauer/oslo/.

[43] Trusted Boo. Available: http://sourceforge.net/projects/tboot/.

[44] McCune J M, Parno B J, Perrig A, et al. Flicker: An Execution Infrastructure for TCB Minimization. In: Proceedings of the 3rd SIGOPS/EuroSys Conference on Computer Systems. Glasgow, Scotland UK, 2008, 315-328.

[45] Goldberg R. Survey of Virtual Machine Research. IEEE Computer Magazine, June 1974, Volume 7:34-45.

[46] Karger P A, Zurko M E, Bonin D W, et al. A Retrospective on the VAX VMM Security Kernel. IEEE Transactions on Software Engineering, 1991, 17 (11):1147-1165.

[47] Security: IBM zSeries partitioning achieves highest certification. Available: http://www-1.ibm.com/servers/eserver/zseries/security/certification.html.

[48] Garfinkel T, Pfaff B, Chow J, et al. Terra: A Virtual Machine-Based Platform for Trusted Computing. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles. Bolton Landing, NY, USA, 2003, 193-206.

[49] Berger S, Cáceres R, Goldman K, et al. vTPM: Virtualizing the Trusted Platform Module. In: Proceedings of the 15th Conference on USENIX Security Symposium, Volume 15:21-21.

[50] 徐明迪, 张焕国, 赵恒, 李峻林, 严飞. 可信计算平台信任链安全性分析. 计算

机学报, 2010,33(7):1165-1176.

[51] Intel Corp. Intel® Trusted Execution Technology. Available: http://www.intel.com /technology/security/.

[52] Microsoft Corp. Next Generation Secure Computing Base. Available: http:// www.microsoft.com/resources/ngscb/default.mspx.

[53] Berger S, Cáceres R, Pendarakis D, et al. TVDc: Managing Security in the Trusted Virtual Datacenter, ACM SIGOPS Operating Systems Review, 2008, 42(1):40-47.

[54] Griffin J L, Jaeger T, Perez R, et al. Trusted Virtual Domains: Toward Secure Distributed Services. In: Proceedings of the 1st IEEE Workshop on Hot Topics in System Dependability. 2005.

[55] Cabuk S, Dalton C, Ramasamy H, et al. Towards Automated Provisioning of Secure Virtualized Networks. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria. Virginia, USA, 2007, 235-245.

[56] Sailer R, Jaeger T, Valdez E, et al. Building a MAC-based Security Architecture for the Xen Open-source Hypervisor. In: Proceedings of the Annual Computer Security Applications Conference. Tucson, AZ, USA, 2005, 276-285.

[57] Brewer D F C, Nash M J. The Chinese Wall Security Policy. In: Proceedings of the 1989 IEEE Symposium on Security and Privacy. Oakland, CA, USA, 1989, 206 -214.

[58] Boeboert W E, Kain R Y. A Practical Alternative to Hierarchical Integrity Policies. In: Proceedings of the 8th National Computer Security Conference. Gaithersburg, MD, USA, 1985, 18-27.

[59] McCune J, Jaeger T, Berger S, et al. Shamon: A System for Distributed Mandatory Access Control. In: Proceedings of the 22th Annual Computer Security Applications Conference. Shanghai, China, 2006, 23-32.

[60] Jaeger T, Sailer R, Sreenivasan Y. Managing the Risk of Covert Information Flows in Virtual Machine Systems. In: Proceedings of the 12th ACM symposium on Access control Models and Technologies. Sophia Antipolice, France, 2007, 81- 90.

[61] Schellhorn G, Reif W, Schairer A, et al. Verification of a Formal Security Model for Multiapplicative Smart Cards. In: Proceedings of the 2000 European Symposium on

Research in Computer Security. Toulouse, France, 2000, 17-36.

[62] 李晓勇, 左晓栋, 沈昌祥. 基于系统行为的计算平台可信证明. 电子学报, 2007, 35(7):1234-1239.

[63] 张晓菲, 许访, 沈昌祥. 基于可信状态的多级安全模型及其应用研究. 电子学报, 2007, 35(8):1511-1515.

[64] Yan F, Qiang W, Shen Z, et al. Daonity: An Experience on Enhancing Grid Security by Trusted Computing Technology. In: Proceedings of the 3rd International Conference on Autonomic and Trusted Computing. Springer, 2006, volume 4158: 227-235.

[65] Jaeger T. Sailer R, Shankar U. PRIMA: Policy Reduced Integrity Measurement Architecture. In: Proceedings of the 11th ACM Symposium on Access Control Models and Technologies. New York, NY, USA, 2006, 19-28.

[66] Armbrust M, Fox A, Griffith R, et al. Above the Clouds: a Berkeley View of Cloud Computing. Technical Report No. UCB/EECS-2009-28, University of California at Berkley, USA, 2009.

[67] Foster I, Kesselman C. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, 1999, 15-51.

[68] Buyya R, Yeo C S, Venugopal S. Market-oriented Cloud Computing: Vision, Hype, and Reality for Delivering It Services as Computing Utilities. In: Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications. 2008, 5-13.

[69] Arenas A. State of the Art Survey on Trust and Security in Grid Computing Systems. CCLRC Technical Report, RAL-TR-2006-008, 2006, 9-21.

[70] Martin A, Yau P. Grid Security: Next Steps. Information Security Technical Report, 2007, 12(3): 113-122.

[71] Jaeger P T, Lin J, Grimes J M. Cloud Computing and Information Policy: Computing in a Policy Cloud? Journal of Information Technology and Politics, 2008, 5(3): 269-283.

[72] Nimbus. Available: http://workspace.globus.org.

[73] Ta-Min R, Litty L, Lie D. Splitting Interfaces: Making Trust between Applications

and Operating Systems Configurable. In: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation. Seattle, WA, USA, 2006, 279-292.

[74] Yang J, Shin K G. Using Hypervisor to Provide Data Secrecy for User Applications on a Per-Page Basis. In: Proceedings of the fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Seattle, WA, USA, 2008, 71-80.

[75] Chen X, Garfinkel T, Lewis C E, et al. Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems. In: Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems. Seattle, WA, USA, 2008, 2-13.

[76] Sahita R, Savagaonkar U R, Dewan P, et al. Mitigating the Lying-Endpoint Problem in Virtualized Network Access Frameworks. Lecture Notes in Computer Science, 2007, Volume 4785: 135-146.

[77] Cox R S, Hansen J G, Gribble S D, et al. A Safety-Oriented Platform for Web Applications, In: Proceeding of IEEE Security and Privacy. Oakland, CA, USA, 2006.

[78] Karger P A, Safford D R. I/O for Virtual Machine Monitors: Security and Performance Issues. IEEE Security & Privacy, 2008, 6(5): 16-23.

[79] Garfinkel T, Rosenblum M. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In: Proceedings of the 10th Network and Distributed System Symposium. San Diego, USA, 2003, 191-206.

[80] Seshadri A, Luk M, Qu N, et al. SecVisor: a Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes. In: Proceedings of the 21st ACM Symposium on Operating Systems Principles. Stevenson, Washington, USA, 2007, 335-350.

[81] Trusted Computer System Evaluation Criteria. Available: http://csrc.nist.gov /publications/ history/ dod85.pdf

[82] 中国信息安全产品测评认证中心．信息技术安全性评估准则．GB／T 18336-2001, (2005-07 -08) . Available: http://www.itsec.gov.cn/

[83] Ogurtsov N, Orman H, Schroeppel R, et a1. Experimental Results of Covert Channel Limitation in One-Way Communication Systems. In: Proceedings of the 1997 Symposium on Network and Distributed System Security. 1997.

[84] Kang M H, Moskowitz I S. A Pump for Rapid, Reliable, Recure Communication. In: Proceedings of the 1st ACM Conference on Computer and Communications Security. Fairfax, VA, USA, 1993, 119-129.

[85] Hu W. Reducing Timing Channels with Fuzzy Time. In: Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy. Oakland, CA, USA, 1991.

[86] Goldschlag D M. Several Secure Store and Forward Devices. In: Proceedings of the Third ACM Conference on Computer and Communications Security. New Delhi, India, 1996, 129-137.

[87] Lin T Y. Chinese Wall Security Policy- an Aggressive Model. In: Proceedings of the Fifth Annual Computer Security Applications Conference. Tucson, Arizona, USA, 1989, 282-289.

[88] Meadows C. Extending the Brewer Nash model to a Multilevel Context. In: Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy. Oakland, CA, USA, 1990, 95 -102.

[89] Sobel A E K, Alves J. A Trace-Based Model of the Chinese Wall Security Policy. In: Proceedings of the 22nd National Information Systems Security Conference. Arlington, Virginia, USA, 1999.

[90] Lin T Y. Chinese Wall Security Model and Conflict Analysis. In: Proceedings of the 24th Annual International Computer Software and Applications Conference. Taiwan, China, 2000, 122-127.

[91] Kessler V. On the Chinese Wall Model. In: Proceedings of the Second European Symposium on Research in Computer Security. Toulouse, France, 1992, 41-54.

[92] Sandhu R S. A Lattice Interpretation of the Chinese Wall Policy. In: Proceedings of the 15th NIST-NCSC National Computer Security Conference. Washington, USA, 1992, 329-339.

[93] 何永忠, 李晓峰, 冯登国. RBAC 实施中国墙模型及其变种的研究. 计算机研究

与发展, 2007, 44 (4) :615-622.

[94] 秦超, 陈钟, 段云所. Chinese Wall 策略及其在多级安全环境中的扩展. 北京大学学报, 2002, 38(3):369-374.

[95] Foley S N. Building Chinese Walls in Standard UnixTM. Unix Computers and Security Journal, 1997. 16(6): 551-563.

[96] Katsuno Y, Watanabe Y, Furuichi S. Chinese Wall Process Confinement for Practical Distributed Coalitions. In: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies. NY, USA, 2007, 225-234.

[97] Lin T Y. Chinese Wall Security Policy Models: Information Flows and Confining Trojan Horses. In: Proceedings of the 17th IFIP11.3 Working Conference on Database and Applications Security. Estes Park, Colorado, USA, 2003, 275-287.

[98] Zhu W, Wang F Y. Covering Based Granular Computing for Conflict Analysis. Intelligence and Security Informatics, Springer Berlin, 2006, Volume 3975: 566-571.

[99] Ao X, Minsky N H. Flexible Regulation of Distributed Coalitions. Computer Security, Springer Berlin, 2003, Volume 2808: 39-60.

[100] Sadeghi A R, Stüble C. Towards Multilateral-Security on DRM Platforms. Information Security Practice and Experience, Springer Berlin, 2005, Volume 3439: 326-337.

[101] Commercial Technology in High Assurance Applications. Available: http://www.vmware.com/pdf/ TechTrendNotes.pdf.

[102] Spencer R, Smalley S, Loscocco P, et al. The Flask Security Architecture: System Support for Diverse Security Policies. In: the Proceedings of the Eighth USENIX Security Symposium. Washington, USA, 1999, 123-139.

[103] 石磊, 金海, 邹德清. Xen 虚拟化技术. 武汉: 华中科技大学出版社, 2009, 320-323.

[104] IDC's Worldwide Server Virtualization Taxonomy, 2010. Available: http://www.idc.com/.

附录 1 攻读博士学位期间发表的学术论文

[1] Ge Cheng, Hai Jin, Deqing Zou, Alex K. Ohoussou, Feng Zhao. A Prioritized Chinese Wall Model for Managing the Covert Information Flows in Virtual Machine Systems. In Proceedings of the 9th International Conference for Young Computer Scientists(ICYCS 2008). Hunan, China, November, 2008, pages 1481 – 1487.(EI)

[2] Ge Cheng, Hai Jin, Deqing Zou, Lei Shi, Alex K. Ohoussou. CFCC: A Covert Flows Confinement Mechanism for Virtual Machine Coalitions. In 3rd International DMTF Academic Alliance Workshop on Systems and Virtualization Management. Wuhan, China, September, 2009.

[3] Ge Cheng, Hai Jin, Deqing Zou, Xinwen Zhang, Min Li, Chen Yu, Gufu Xing. Building Dynamic Integrity Protection for Multiple Independent Authorities in Virtualization-based Infrastructure. In Proceedings of the IEEE/ACM International Conference on Grid Computing (Grid 2009). Banff, AB, Canada, October, 2009, pages 113-119.(EI)

[4] 程戈, 金海, 邹德清, 赵峰. 基于动态联盟的中国墙模型研究. 通信学报, 2009, 39(11):93-101. (EI)

[5] Ge Cheng, Hai Jin, Deqing Zou, Xinwen Zhang. Building Dynamic and Transparent Integrity Measurement and Protection for Virtualized Platform in Cloud Computing. Journal of Concurrency and Computation: Practice and Experience, Wiley. 2010,22(13) :1983-1910 (SCI)

[6] Alex K. Ohoussou, Hai Jin, Deqing Zhou, Guofu Xiang, Ge Cheng. Autono. Autonomous Agent Based Intrusion Detection in Virtual Computing Environment. In 2010 IEEE International Conference on Wireless Communications, Networking and Information Security (WCNIS2010). Beijing, China, June 2010, pages 682- 686 (EI)

附录 2 攻读博士学位期间申请发明专利与软件著作版权

申请国家发明专利

[1] 金海，程戈， 邹德清，赵峰, 石磊：基于优先中国墙策略的虚拟机隐形流控制方法。 专利申请号：200810047946.8， 申请日期： 2008 年 6 月 6 日，申请单位：华中科技大学。

[2] 金海，程戈，邹德清，羌卫中，余辰：一种基于虚拟机架构的透明信任链构建系统。专利申请号：201010214334.0，申请日期：2010 年 6 月 30 日，申请单位：华中科技大学。

申请国家软件著作版权

[1] 虚拟化云计算环境下的完整性度量系统，版本号：V1.0. 软件著作版权登记号：2010SR020957，登记日期：2010 年 5 月 9 日，申请单位：华中科技大学。

附录 3 攻读博士学位期间参加的主要科研项目

[1] 国家重点基础研究发展计划（973 计划）项目, 计算系统虚拟化基础理论与方法研究（No.2007CB310900），2007.7-2011.12。

[2] 国家自然科学基金资助项目，逻辑虚拟域中软件执行的可信确保机制研究（No.60973038）， 2010.1-2012.12。

[3] 武汉市科技攻关项目，基于云计算的网络安全防御体系研究，（No.201010621211），2010.1-2011.12。

附录 4 个人简历

基本信息

姓　　名：程 戈　　性　　别：男

院　　校：华中科技大学计算机学院　　专　　业：计算机体系结构

电话号码：15173205270　　政治面貌：中共党员

E-mail：　chenggecn@gmail.com　　导　　师：金 海 教授

研究兴趣

可信计算、虚拟化、云计算

教育背景

2007.09-今	华中科技大学	计算机学院集群与网格计算实验室	博士生
2002.09-2005.07	湘潭大学	数学与计算科学学院	硕士生
1996.09-2000.07	湘潭大学	数学与计算科学学院	本科生

实践经历

2000.09-至今	湘潭大学	数学与计算科学学院	教师

所获奖励