附件1.2：学术型硕士学位论文封面格式（浅黄色A4皮纹纸装订，打印时此行取消）

分类号： (按中国图书分类法) 单位代码： 10636

密 级：（注明密级与保密期限） 学 号：



**硕士学位论文**



**中文论文题目 ：** （小二号仿宋体加黑）

**英文论文题目：** （16pt Time New Roman，Bold）

论文作者： （四号仿宋）

指导教师：

专业名称：

研究方向：

所在学院：

论文提交日期： 年 月 日

论文答辩日期： 年 月 日

附件2：论文独创性及使用授权声明

四川师范大学学位论文独创性声明

本人声明：所呈交学位论文 ，是本人在导师 指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

本人承诺：已提交的学位论文电子版与论文纸本的内容一致。如因不符而引起的学术声誉上的损失由本人自负。

学位论文作者： 签字日期： 年 月 日

四川师范大学学位论文版权使用授权书

本人同意所撰写学位论文的使用授权遵照学校的管理规定：

学校作为申请学位的条件之一，学位论文著作权拥有者须授权所在大学拥有学位论文的部分使用权，即：1）已获学位的研究生必须按学校规定提交印刷版和电子版学位论文，可以将学位论文的全部或部分内容编入有关数据库供检索；2）为教学、科研和学术交流目的，学校可以将公开的学位论文或解密后的学位论文作为资料在图书馆、资料室等场所或在有关网络上供阅读、浏览。

本人授权万方数据电子出版社将本学位论文收录到《中国学位论文全文数据库》，并通过网络向社会公众提供信息服务。同意按相关规定享受相关权益。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名： 导师签名：

签字日期： 年 月 日 签字日期： 年 月 日

附件3：论文摘要格式

高中语文古诗词鉴赏教学研究

□□□□□□专业

研究生 □□□□ 指导教师 □□□□

**摘要** 古诗词是中华民族文化中的精华。古诗词鉴赏教学可以激发学生的情感，提高学生的审美情趣、陶冶学生的情操，使学生受到传统文化、人文精神、道德修养、文化品味的熏陶和感染，这对于继承民族文化传统，弘扬民族精神、实现语文的美育功能都具有非常重要的意义。

本文分四章来论述高中语文古诗词鉴赏教学，……

本文将从知人论世策略；比较策略；整体感知策略；张扬主体、多元解读策略；营造情景策略五个方面对高中语文古诗词鉴赏教学作探讨。

关键词：高中语文 古典诗词 鉴赏教学

附件4：目次页格式

**目 次**

摘要（小4号宋体） I

ABSTRACT II

插图和附表清单 Ⅲ

目次

1 （第1章）引言（绪论） 1

1.1（第1章第1节）题名 3

2 （第2章）题名 5

2.1（第2章第1节）题名 7

2.2（第2章第2节）题名 10

…………

5 （第5章）结论 71

参考文献 93

附录 96

致谢 98

在校期间的科研成果 102

**1** 绪论

**1.1 研究背景及意义**

根据NIST定义，云计算是可以使用户按照使用量付费并且获得高效的、快捷的网络服务的新型资源共享模式，其主要目的是在提高网络、服务器、硬盘存储、软件等共享资源利用率的同时，使云租户不再关注硬件资源的管理和维护，云租户只需要在硬件资源上投入很少的管理维护工作就可以得到很高的资源回报。云计算的高效的资源处理能力也带动了大数据、人工智能等相关领域的发展。目前在云计算提高商，比如国外的Intel，IBM，微软，以及国内的腾讯，阿里巴巴都拥有非常成熟的云计算技术和应用服务提供技术。云计算的快速发展同时也给云计算带了除传统信息安全、网络安全之外的安全问题，其中，如何向云租户证明云计算底层平台的安全性、虚拟机的安全性是一个非常重要的问题。而可信计算是保障信息系统安全最为重要的技术手段之一，它通过提供数据保护、身份认证、远程证明以及完整性度量等特性提高包括底层物理资源、应用软件等在内的计算平台的可信性和可靠性。因此，将可信计算技术应用在提高云计算环境的安全性是工业界和产业界必须重视的地方。

可信平台模块TPM(Trusted Platform Module, TPM)是可信计算的核心和关键技术，是可信计算机系统的信任根，通过构建从平台底层硬件到平台上层应用程序的信任链，并结合可信远程证明向平台外部实体提供可信证明。TCG（Trusted Computer Group, TCG）定义的可信计算平台的核心功能：度量、存储和报告等均依赖于TPM；可信计算平台的3个信任根：可信测量根RTM(Root of Trust for Measurement, RTM )、可信存储根RTS(Root of Trust for Storage, RTS )和可信报告根RTR(Root of Trust for Report, RTR)等均与TPM有直接的关系，其中可信测量根RTM由CRTM(Core Root of Trust for Measurement, CRTM)和TPM中的一组PCR存储器组成，可信存储根RTS由TPM和存储根密钥SRK组成，可信报告根RTR由TPM和EK组成。TCG的TPM工作组负责TPM规范的起草、修订和发布，其规范已从TPM1.0、TPM1.2发展到现在的TPM2.0。TPM2.0规范于2014年发布，在密码算法支持、密钥、授权、签名、虚拟化等方面均有些新的特点，尤其在虚拟化方面的更新推动了可信计算技术与云计算技术的结合，共同保障云计算安全，向云租户提供一个安全可靠的云服务。利用可信计算技术构建可信虚拟平台（TVP）是研究的重要方向。TVP可以利用可信计算技术对云计算平台启动过程中，不仅可以对云计算物理资源组件进行信任链可信度量，比如BIOS、操作系统内核OS、虚拟机管理器等，也可以对虚拟机启动的组件进行可信度量，比如虚拟机镜像系统、虚拟机文件、虚拟机虚拟化过程所需的OS、虚拟BIOS等。TVP的出现能够帮助云计算提供商向云租户更好的提供安全可靠的云服务。

根据中国信息通信研究院2017年7月发布的《云计算关键行业应用报告》，2016 年以 IaaS、PaaS 和 SaaS 为代表的典型云服务市场规模达到 654.8 亿美元，增速 25.4%，预计 2020年将达到 1435.3 亿美元，年复合增长率达 21.7%。我国云计算市场总体保持高速增长趋势。2016 年我国云计算整体市场规模达 514.9 亿元，整体增速 35.9%，高于全球平均水平。工业和信息化部发布的《云计算发展三年行动计划（2017－2019年）》提到我国云计算的发展目标“到2019年，我国云计算产业规模达到4300亿元”，该行动计划为我国云计算安全技术创新和产业发展指明了方向，提供了政策保障和法律依托。并且根据McAfee发布的“2017年全球云计算安全报告”显示，在过去15个月，80%的IT预算用于云应用和解决方案。云计算已经成为通过部署应用来销售服务的企业的首要事项。很多企业正在快速向云计算转移，以跟上合作伙伴和供应商的网络。无论顾客选择任何购买渠道和方式，都能提供卓越的客户体验，这也推动了云计算的部署。但是，只有23%的企业完全信任公共云可以保护他们的数据安全。由此可见目前的大部分云租户是不能完全信任云提供商提供的云计算服务，主要是由于云租户在使用云提供商提供的虚拟机时，并不能确认云计算平台上的物理主机是与云提供商按照各自操作系统官方文件进行启动的，以及租户请求的虚拟机是按照预期的配置和要求进行启动的。因为云计算环境下的虚拟机存在着包括传统信息系统安全以及新型网络安全等威胁，比如：虚拟机恶意代码攻击、虚拟机逃逸等，这些都会导致虚拟机在重新启动时的组件被篡改，在云租户对虚拟机进行重新启动时，可能无法判断虚拟机遭受操作系统、数据是否被篡改。而可信虚拟平台的构建可以利用TPM中的可信度量、可信报告等技术向用户发送关于云计算平台的可信度量结果，并且证明自身的安全性。

信任链技术是可信计算的关键技术，针对可信计算技术与云计算技术结合的可信虚拟平台的信任链构建更是十分有必要的。利用虚拟可信平台模块（vTPM）在云计算环境中构建安全可靠的可信虚拟平台，并且利用可信计算中的关键技术——信任链技术对整个云计算平台进行信任链构建，建立从云计算平台底层物理服务器到提供服务的可信虚拟机的信任链，可以给目前的云计算安全问题提供一个新的解决思路，为构建云服务提供更好的安全保障。将可信计算技术与虚拟化技术相结合构建虚拟可信计算平台具有的优势可以从两方面说明。一方面，虚拟化技术可以对可信计算平台提供安全支持。通过VM11」提供隔离和监控机制，为在操作系统和软件应用层建立可信计算环境提供了保漳，有效防止外界对系统或程序的侵扰;另一方面，可信计算平台为虚拟化技术提供安全保障服务，并将可信计算融入基于虚拟机技术的应用业务当中，提高了应用及数据的安全性。

为此，本文对可信虚拟平台及其之上的信任链进行研究，并利用已有的形式化分析方法安全逻辑方法和改进的无干扰理论分别对信任链进行形式化分析。首先提出了带有可信衔接点的可信虚拟平台TVP-QT，该可信衔接点由三部分组件构成，虚拟机构建模块，虚拟可信模块构建模块，VM和vTPM绑定模块，该可信衔接点充分连接了云计算平台和虚拟机，保障信任链构建的合理性和完整性。其次，本文提出了一种具有瀑布特征的信任链模型—TVP-QT，该模型以硬件TPM为起点，在底层虚拟化平台和顶层用户虚拟机信任链之间加入可信衔接点。当信任链从底层虚拟化平台传递到可信衔接点时，由可信衔接点负责对用户虚拟机的vTPM进行度量，之后将控制权交给vTPM，由vTPM负责对用户虚拟机启动的组件及应用进行度量。该模型中可信衔接点具有承上启下的瀑布特征，能满足虚拟化环境的层次性和动态性特征，保证了整个可信虚拟平台的可信性。然后，本文利用目前的安全逻辑形式方法和改进后的无干扰理论方法——无干扰理论+方法对TVP-QT进行形式化分析，不仅在实验上证明了TVP-QT的合理性和可靠性，也在理论上证明了TVP-QT的合理性。

**1.2 国内外研究现状**

可信计算技术与虚拟化技术的结合的TVP一直以来都受到国内外学者的广泛关注。目前已经有涌现出很多优秀的研究成果。Intel的StefanBerger等人最先提出vTPM的概念，随后产生了很多关于TVP及其信任链构建的研究成功，其中，开源的VMM——Xen是最早支持vTPM的，已经有许多学术成果将Xen作为实验平台；KVM对支持vTPM主要是依赖QEMU；2015年，VMware宣布在vSphere 6.0中支持vTPM；Oracle的Virtual Box也通过IBM的PCIXCC支持vTPM。这些知名公司的技术促进了可信虚拟平台的发展。为更好的对国内外研究现状进行阐释，本文将从可信虚拟平台、可信虚拟平台信任链、针对信任链的形式化分析方法三部分进行描述。

1.2.1 **可信虚拟平台**

对于TVP的研究，早在TVP概念出现之前，就出现了利用可信计算技术解决虚拟系统平台安全的方案，为TVP的发展提供了一些理论和构建基础，这些平台包括Terra [17]，Perseus [18]等，这些平台的主要思想是把底层计算平台分为两部分，可信区域和不可信区域，其中可信区域上运行着高安全性需求的虚拟机。

TVP 的概念首先由 Stefan Berger [12] 等人提出，随后文献[13-16]等学者针对如何构建具体应用场景的 TVP 功能应用以及抽象和统一的 TVP 概念取得了很多较好的研究成果，并且达成了一些基本的共识。目前，研究此方面的学者绝大多数都认为，在物理上，TVP作为一个可以支持虚拟化技术的可信主机，并且与一般的可信计算平台的主要区别有两方面，一是拥有在物理硬件可信平台模块（Trusted Platform Module, TPM）构建起来的虚拟可信信任根；而是可以并发的为在可信虚拟平台之上的多个用户虚拟机（Virtual Machine, VM）提供可信虚拟信任环境。这种TVP的运行架构如图1所示。从功能上看，TVP架构主要分为4个层次。第一层为硬件信任根TVP，作为整个架构的最底层，是整个平台信任的物理保证。第二层主要包括虚拟机监视器（Virtual Machine Monitor, VMM），及构建与VMM之上的管理域（主要是其内核及相关域管理工具），它们通常被认为是 TVP 的可信计算基（Trusted Computing Base, TCB）。第三层是虚拟信任根（Virtual Root of Trust, vRT），由于实现方案不同（如图1中a、b所示），其加载过程可能是传统信任链的一部分，或直接利用动态加载机制如动态度量信任根（Dynamic Root of Trusted Measurement, DRTM）机制启动，这使得它或者成为 TCB 的一部分，或者作为应用进程单独存在。最上层是用户虚拟机，是与用户应用密切相关的部分。



图1TVP基本运行架构

根据文献[12]的vRT等概念，HP、IBM等研究机构分别提出并构建了相应的TVP[13][14]，其TVP架构可根据不同应用需求建立用户可定制的TVP，在很大程度上推动了TVP的发展。随后，Krautheim[15][[[1]](#endnote-0)][[[2]](#endnote-1)[[[3]](#endnote-2)]、王丽娜[16]等学者基于云计算环境建立了TVP，使其可以保护云计算环境下的虚拟机运行，以及保护虚拟机运行时上层服务软件的完整性、安全性。之后，常德显[19]等根据TVP的功能层次给出了包括虚拟机和虚拟可信根的TVP定义，并细分为VMM、Dom0、TPM、vRT等组件。Zhang Lei[20]等提出一种具有可信域层次的TVP，通过可信云平台和可信虚拟机进行分离的TVP构建机制，并实现了对可信云平台以及可信虚拟机的安全保障。文献[21-26]也建立了类似以上的可信虚拟平台。可以说TVP在保证云计算环境安全、构建可信云平台上起到了重要的作用。总结起来，目前针对TVP模型的研究尽管取得了很多成果并达成了基本共识，即TVP模型都包含基本组件vRT、vTPM等，但绝大多数已有的研究成果把TVP的VMM和管理域都作为TCB，一起作为虚拟机的vRT，这显然过粗且逻辑上不完全合理的，因为管理域包含OS及大量的应用程序，显然不能采用链式度量所有的应用程序并存储其PCR值。

上述的TVP基本运行架构以及信任链传递模型存在过粗且逻辑上不完全合理的问题，与具体云环境中虚拟化平台也不完全相符合。如图1所示，为了便于叙述，本文将图1中从TPM到第三层的信任链称为可信虚拟平台信任链，将第四层的信任链称为虚拟机信任链。具体问题表现在：

（1）现有的TVP模型把整个第三层都作为TVP的TCB并作为虚拟机的vRT，显然是不精细的且逻辑上也不完全合理的。第三层包括VMM以及DOM管理域，信任链为CRTM→BIOS→BootLoader→VMM→DOM OS→Apps，DOM管理域包含OS及大量的应用程序，显然不能采用链式度量所有的应用程序并存储其PCR（Platform Configuration Module）值。

（2）虚拟平台信任链与虚拟机信任链是两条不同的信任链，即在整个TVP以及客户虚拟机启动过程中存在两条完全分隔的信任链，一条是可信虚拟平台在启动时的信任链，另一条是客户虚拟机在启动时的信任链，这两条信任链在度量层次和度量时间上均是分离的。如何向虚拟机用户展示一条从TPM到虚拟机应用的完整信任链呢？显然，这两条信任链存在如何衔接的问题。

1.2.2 **可信虚拟平台信任链模型**

对于TVP信任链模型的研究，主要包括三个方面。其一是通过对TCG链式信任链模型的扩展，实现TVP下可信度量以及信任传递。Scarlata[27]等提出在构建TVP时，通过可信测量构建从CRTM可信根到每个客户虚拟机的信任链，就可以证明每个客户虚拟机是可信的，显然这种信任链模型是不完善的，无法适应比较复杂的TVP环境。John[28]对信任链扩展上提出了“Transitive Trust Chain”信任链模型，并且简要的指出了信任链传递过程为TPM →VMM→TVEM manager →TVEM→VM OS(应用程序，但是此种信任链模型没有详细的描述特权域操作系统以及虚拟机操作系统的可信度量。Shen[29]等根据TCG动态度量方法提出了一种基于Xen的可信虚拟机在 DRTM下的信任链构建，其具体的构建过程为：CPU→可信代码→Xen VMM→Dom0（→vTPM Manager→Domain Builder）→Guest 0S →Guest Application，此种信任链模型也存在John[28]中的问题。等等。其二是通过研究可信云平台和可信虚拟机两部分的信任链，构建TVP下的信任链模型。常德显[19]等提出TVP信任链包括按照TVP的功能层次从硬件TPM层→TCB层→vRT层→用户虚拟机层的信任链模型，此信任链模型对vRT及层次间的连接定义比较模糊。Zhang Lei[20]等提出一种基于无干扰的可信域层次信任链模型，并且指出分别度量物理主机和VM的方式，即首先度量从物理的TPM到物理主机的应用程序，然后度量VM的vTPM和应用程序，显然此种信任链模型无法有效的对TVP下构建完整的链式信任链模型，不能向用户虚拟机呈现一条完整的信任链模型，文献[22][23]也存在此类问题。其三是树形或者星形的信任链模型。其三是树形或者星形的信任链模型。一部分学者认为TCG的链式信任链可信度量方式在虚拟化环境下是难以有效构建的。朱智强[30]提出了一种安全可扩展的星型信任度量结构，在信任度量时只需要信任根（RT）对管理域节点进行度量即可，但是此信任链模型的关键节点RT需要对所有的管理节点进行度量，RT的负担重，无法高效的完成TVP下的可信度量以及信任传递。曲文涛等[31]提出了一种解决RT负担的改进方案，带链式结构的星型信任链模型，设计了MDn节点分担了RT的部分度量负担，但是此种信任链模型也存在负担重的MDn节点。总结起来，目前针对TVP的信任链模型的共同问题是信任链模型过粗且逻辑上不合理的问题，与具体云环境中虚拟化平台也不完全相符合，且目前研究内容中的可信虚拟平台信任链与虚拟机信任链是两条不同的信任链，这两条信任链在度量层次和度量时间上均是分离的，不能向虚拟机用户展示一条从TPM到虚拟机应用的完整信任链。

1.2.3 **无干扰理论**

针对于确保平台信任可验证的信任链形式化建模与分析的方法，目前的研究大部分是基于传统的可信计算平台。陈书义[21]等人利用一阶逻辑对可信计算平台启动过程进行建模以分析其信任传递过程，并提出长度受限的信任链模型。张兴[22]等人基于无干扰模型对信任链进行了建模分析，从系统信息流控制角度验证满足传递无干扰安全策略的信息流才能构建有效的信任链。上述方法主要针对普通可信计算平台，并不能直接适用于云计算环境下信任链形式化分析。虽然Zhang[30]等人利用无干扰理论对可信云计算环境信任链进行了形式化分析和验证，但是此种信任链分析方法是建立在不连续的可信云计算信任链模型上，不能够对可信云计算环境进行正确的形式化验证。常德显[29]等人基于扩展安全系统逻辑的分析方法也存在Zhang中的问题。

针对于无干扰理论的研究，目前大部分的研究是基于信息流的无干扰模型从动作和运行结果的角度建立系统安全策略模型。张兴、赵佳[33,34]等在Rushby[35]的无干扰理论的基础上将系统安全域集实体化为进程集，给出了进程运行的可信条件，推导出系统运行可信定理，保证了终端的安全，但是其模型中的没有针对动作的详细定义，不适合验证可信云环境信任链。刘鹏威[36]等提出了基于非传递的无干扰理念的二元多级安全模型，在Rushby无干扰理论的基础上重新定义了清除函数，将传递的元干扰理论过渡到非传递的无干扰理论，并依据BLP和Biba模型保护了信息的机密性和完整性，然而同样存在赵佳中的问题。陈菊[37]等从进程数据和代码完整性检测出发，利用无干扰理论保证进程之间的操作合法，试图在不安全的操作系统中建立安全的应用支撑。徐甫[38]等扩展了非传递无干扰理论，并试图通过重新定义静态干扰和动态干扰，使其支持进程自身代码的修改，但是其静态干扰和动态干扰的定义过于抽象，难以和实际的终端系统相对应，因此并不能在实际上完成其所描述的支持自身代码修改的功能。秦晰[39]等提出了一种容忍非信任组件的可信终端模型，该模型利用可信组件对非信任组件的输出进行封装，保证了非信任组件在终端上的存在不会造成严重的安全威胁，实现了域间隔离和无干扰，保证了结果的可预测性和可控性，但是并没有针对安全域进行详细的描述。上述对无干扰理论的研究，均没有考虑到云计算运行中时的安全域、动作所属主体以及动作对安全域和系统状态的影响进行详细的说明，比如针对于安全域，在运行中各个安全域可能是独立也可能是通过组件绑定在一起。

**1.3 本文主要工作**

1. 具有瀑布特征的可信虚拟平台

由本文对可信虚拟平台的研究综述来看，目前的可信虚拟平台存在一些需要改进的地方。本文针对现有的可信虚拟平台，在可信虚拟平台和用户虚拟机之间加入可信衔接点，设计一种在逻辑上合理，vRT精细的可信虚拟平台。此可信虚拟平台的主要架构如下图所示：

在原有的TVP中的第二层和第三层加入可信衔接点。可信连接点位于Dom0，是Dom0的一组应用程序，包括VM的创建组件VM Builder、 vTPM实例的创建模块vTPM Builder以及VM-vTPM映射组件VM-vTPM Binding，且作为vRT的一部分，在信任链上按照VM BuildervTPM Builder VM-vTPM Binding的顺序依次进行度量。可信衔接点可对TVP-QT的第一、第二层与第四、第五层进行有效衔接，保证TVP-QT信任链构建的连贯性，起到承上启下的作用，具有瀑布特征。所谓“瀑布特征”，借鉴软件开发过程中的“瀑布模型”，软件开发中的“瀑布模型”，其特点是每一个开发流程必须从上一项开发流程接收输入，并且为下一个流程产生输出；如果重点任何一个环节没有输入或者产生输出，则严重影响开发的流程，甚至终端。本文提出的具有可信衔接点的TVP-QT模型在任何一个环节都会与前后组件进行衔接，如果中间因任何不安全问题出现任何错误或者中断，则TVP-QT及其信任链构建都会终止。

其中VM Builder表示与创建用户虚拟机相关的配置文件以及组件等，如创建vm组件以及vm配置文件等；vTPM Builder表示与创建和管理vTPM实例的相关组件，并负责提供给vm运行时的vTPM标识以及端口的提供。而VM-vTPM Binding则表示对vm和vTPM实例间的绑定关系，在TVP-QT涉及到的vTPM架构中，由vTPM Builder提供给vTPM实例的标识，每个vm必须与独一的vTPM实例绑定。

从此架构图可以看出，可信衔接点的存在可以有效的对用户虚拟机和可信虚拟平台进行衔接，并且具有承上启下的作用，使得整个可信虚拟平台在逻辑上都非常合理，并且加强了用户虚拟机和可信虚拟平台之间的联系。

1. 具有瀑布特征的信任链模型

由本课题针对可信虚拟平台之上的信任链模型，对本课题设计的带有可信衔接点的可信虚拟平台上进行构建，并对可信衔接点部分的组件进行细分，使得整个可信虚拟平台在启动过程中能够对虚拟平台TCB层度量之后能够有效的把控制权交给用户虚拟机的vRT，实现向用户展示一条完整的信任链模型的目标。

本课题构建的信任链模型大体架构如图：

具体的信任链度量过程描述如下：

针对可信虚拟平台构建的信任链模型总体过程如下：

第一层硬件TPM（CRTM）-->第二层TCB（BIOSOSLoaderVMMDom0 Kernel）-->第三层可信衔接点（VM BuildervTPM BuilderVM-vTPM Binding）-->第四层vTPM（vTPM实例）-->第五层可信虚拟机（VBIOSVOSLoaderVMOSAPP）;并利用通过TVP平台对信任链进行实例系统的分析和构建。本课题拟基于Xen VMM对信任链进行构建和实现。

1. 基于扩展无干扰理论的可信虚拟平台信任链分析方法

目前大部分的研究是基于信息流的无干扰模型从动作和运行结果的角度建立系统安全策略模型，本课题按照云计算环境运行特征，拟对原有无干扰理论中的安全域、动作等定义进行扩充，并将动作主体和动作对安全域以及系统状态的影响等扩展到无干扰理论中；最后应用此扩展的无干扰理论来分析可信云环境信任链传递模型，用形式化的方法证明当符合非传递无干扰安全策略时，云环境安全域之间的信息流受到安全策略限制，隔离了域之间的干扰，满足此条件时用完整性度量方法所建立的云环境信任链才是可信的、有效的。

图4 扩展无干扰理论

**1.4 论文组织结构**

本文共分为六章，每部分的安排如下：

第1章主要介绍了论文的研究背景及意义，主要从目前国内外云计算的发展及云计算安全的发展；然后介绍了国内外的研究现状，主要从可信虚拟平台，信任链模型，形式化分析方法三个方面进行介绍；最后介绍了本论文的研究方向和研究内容。

第2章主要介绍了云计算中的关键技术——虚拟化技术，以及虚拟机和虚拟机监视器，目前最流行的VMM结构Xen和KVM；其次介绍了可信计算中的关键技术，主要介绍了虚拟可信平台模块，信任链的构建和完整性度量技术，以及虚拟可信平台模块的分类和目前信任链技术的不足；最后介绍了形式化方法，并详细介绍了安全逻辑和无干扰理论形式化分析方法。

第3章主要介绍了本文提出的TVP-QT可信虚拟平台，从TVP-QT的工作流程到实现。首先提出了基于可信衔接点的TVP-QT，可信衔接点由虚拟机构建模块，vTPM构建模块，VM-vTPM绑定模块组成，不仅可以当做可信云平台底层物理服务器的一部分也可以当做虚拟机启动的一部分，完成对底层物理和虚拟机的衔接。其次，为方便本文第5章的形式化分析，对TVP-QT的每个部分进行定义。然后，基于Xen实现了TVP-QT，对该可信虚拟平台的每个部分进行仔细的描述，主要包括五个部分。最后对TVP-QT进行了功能试验和性能试验。

第4章主要介绍了构建在TVP-QT之上的信任链模型。首先对Linux的启动过程以及虚拟机的启动过程进行分析，根据可信计算的完整性度量技术，确定了TVP-QT在系统启动过程中物理服务器和虚拟机需要度量的内容，利用Trusted GRUB实现对信任链的构建。其次，对TVP-QT进行信任属性的定义，主要包括本地属性和远程属性。最后对信任链进行了功能和性能试验。

第5章主要利用扩展安全逻辑和扩展无干扰方法对TVP-QT进行形式化分析。其中本文提出的扩展无干扰理论方法可以适用目前的可信虚拟平台信任链的分析。

第6章主要是对本文的研究内容进行概括分析，并指出了论文的不足之处，对未来的研究进行了展望。

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

**2** 相关技术

**2.1 虚拟化技术**

虚拟化技术利用单一硬件模拟多个同类设备，从而提高设备利用率，降低能耗，并支持多个操作系统或应用程序的并行执行。从系统安全的角度出发，虚拟技术可为操作系统或应用程序提供运行时隔离机制，有效防止外界对系统或程序侵扰，有效的保护系

统中数据的安全。

××××××××××××××××××××××××××××××××××××××××××××××

2.1.1 **虚拟化技术分类**

为了满足不同的功能需求，有许多不同种类的虚拟化解决方案。由于采用了不同的实现方式和抽象层次，因而这些虚拟化系统呈现出不同的特性[85-87]。

1. 根据虚拟化技术的实现层次分类

指令级虚拟化：也称为指令集架构级（Instruction Set Architecture，ISA）虚拟化，即以软件模拟出实际运行的操作系统的指令集来实现的方式。模拟器可以将用户虚拟机所产生的指令进行转化从而形成一种本地的指令集，随后将这种本地指令集在计算机上真实的硬件上执行。指令级虚拟化系统主要有 Bochs 系统和 QEMU 系统。

硬件级虚拟化：这种虚拟化与前面所讲述的指令集架构虚拟化存在相似之处，他们之间存在的差别在于这种虚拟化仅仅是对少数的特殊情况进行考虑，这种情况下用户的使用环境与计算机具有相同的指令集，这就为绝大多数的用户指令直接在主机上进行操作提供了便利，提高了指令的运行的速度。这种技术还能够实现虚拟资源的物理化转变，当映射为物理资源以后就可以在虚拟环境下使用计算机的本地硬件。这种虚拟化是当前在计算机领域研究较多的内容，也是虚拟化技术研究中的核心。在虚拟化研究中出现了VMware 和 Xen 等具有影响力的硬件级虚拟化系统。

操作系统级虚拟化：在传统的计算机操作系统中每一位计算机用户的进程本质上是在同一个操作系统的实例中运行，所以操作系统的内核缺陷或者是在程序设计上存在问题都可能会对其他进程产生影响。而对操作系统进行虚拟化则是一种有效解决上述问题的办法，这是一种轻量级的虚拟化技术，在这一技术下可以创建多个虚拟化的操作系统提供给不同的用户，这样每个用户就可以在相对独立的环境下进行操作，从而避免了同一个系统中的各个进程的相互影响，从而导致系统的整体运行效率下降以及出现相互的影响的状况。当前典型的系统有 Linux-VSerser、Open VZ、Solaris Container，Free BSD Jail等。 ××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

1. 根据实现方式不同分类

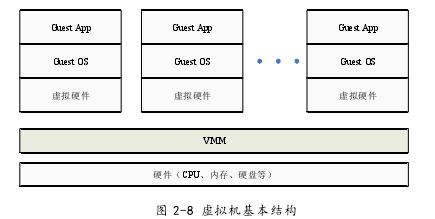
全虚拟化：在全虚拟化的环境下计算机的虚拟机监视器能够虚拟出如同真实硬件环境一样的虚拟硬件环境。因此，每一个虚拟机都可以使用一个虚拟的硬件环境，这个虚拟硬件环境能够提供如同真实的硬件环境相同的支持服务;而在这个支持服务提供中系统不需要真实的硬件或者是操作系统来协助，因此不会对系统的Guest OS内核产生任何的修改，该内核也无法感知到系统中是否发生了虚拟化。

半虚拟化：这是一种与全虚拟化相辅助的一种虚拟化技术，这种虚拟化技术需要借助操作系统的协助才能完成对特权指令的虚拟化，所以会对客户操作系统的内核进行修改，以便操作系统能够发现有问题的指令进行更改。

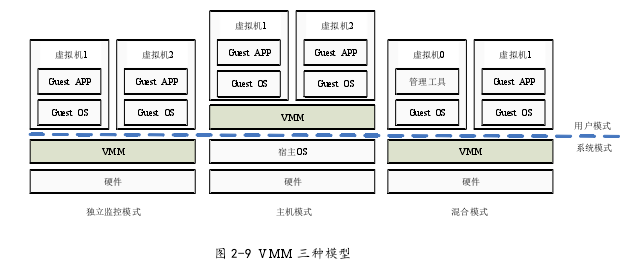
硬件辅助虚拟化：硬件辅助处理敏感指令以实现完全虚拟化的功能，客户操作系统无需修改。这种方案最早由全球两大芯片供应商最先提出的，其基本思想就是引入新的处理器和指令运行模式，使VMM和Guest OS运行在不同的模式下。Guess OS只能在受控的模式下运行，当需要由V卜IM进行监控和模拟时，由硬件支持模式切换。

2.1.2 **虚拟机与虚拟机监视器**

虚拟机(Virtual Machine VM)是一种基于计算机硬件平台工作的虚拟硬件系统，这一系统能够对平台上的ISA结构和实际硬件进行模拟。在这一模拟的硬件系统下每一个不同的系统都可以支持操作系统的运行，这些客户操作系统则是利用虚拟机监视器来实现对实际资源的利用。虚拟机基本结构如图2-8所示。



从这个虚拟机的体系结构来看，vn}是一个位于计算机硬件和操作系统之间的软件层，它运行在系统的特权级，对上层的多个虚拟机进行隔离管理从而保证这些虚拟机能够在一个安全的环境下独立的运行。此外，它还可以为每一个虚拟机提供在计算机中虚拟出的和真实硬件环境无关的虚拟硬件环境。这样，vn}可以将存在于传统构架上的单一程序运行到计算机虚拟出来的虚拟机上，并且可以实现将多个虚拟机的整合到一个物理平台上，降低了系统的管理复杂程度，提高了管理的效率和节约了管理成本，并降低了对计算机的空间资源的占用。研究人员普遍认为VMM作用不只是多任务的管理作用，也是一种更加安全和更加可靠的系统问题解决方案。根据v}所处的物理系统中的位置将其分为独立监控模式、宿主模式和混合模式三种类型，结构如图2-9所示。



××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

2.1.3 **Xen与KVM**

（1）Xen

Xen是英国剑桥大学计算机实验室主持的开源虚拟化项目，Xen项目始于2002年，起初Xen是基于X86体系结构(32位)设计开发的，其设计的目标是希望能够实现对多个虚拟机操作系统的运行提供支持。Xen位于计算机的硬件和操作系统之间，可以向在其上层运行的操作系统内核提供一种虚拟化的计算机硬件模拟环境。可以在一套物理硬件上安全地执行多个虚拟机}。Xen采用混合模式，在虚拟域(Domain, Dom)中有一个特权虚拟域(Domain 0,Dom 0 )，用来辅助Xen管理其它虚拟域(Domain U, Dom U)，为其提供相关的虚拟资源服务。Xen体系结构如图2-10所示。××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

Xen向Domain提供一个抽象层(包含管理和虚拟硬件的API)。其中，Dom。拥有真实的设备驱动(Dative Device Driver,原生设备驱动)，能够直接访问物理硬件，其它的Dam都不能直接访问物理硬件，它们对物理硬件的访问必须通过Dam 0。它负责与Xen提供管理的API进行交互，并通过用户模式下的管理工具管理Xen的虚拟机环境，负责启动、停止和迁移其它的虚拟域，并通过控制接口控制其它虚拟域的CPU调度、内存分配，以及外部设备访问。

（2）KVM

Kernel-based Virtual Machine的简称，是一个开源的系统虚拟化模块，自Linux 2.6.20之后集成在Linux的各个主要发行版本中。它使用Linux自身的调度器进行管理，所以相对于Xen，其核心源码很少。KVM目前已成为学术界的主流VMM之一。KVM的虚拟化需要硬件支持（如Intel VT技术或者AMD V技术)。是基于硬件的完全虚拟化。而Xen早期则是基于软件模拟的Para-Virtualization，新版本则是基于硬件支持的完全虚拟化。但Xen本身有自己的进程调度器，存储管理模块等，所以代码较为庞大。广为流传的商业系统虚拟化软件VMware ESX系列是基于软件模拟的Full-Virtualization。

2.1.4 **本章小结**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

**3** 具有瀑布特征的可信虚拟平台TVP-QT

**3.1 TVP-QT系统结构**

我们基于已有的TVP研究方案，提出了TVP-QT运行架构，如图2所示。



TVP-QT运行架构

本文提出的TVP-QT运行架构在功能上可以分为5个层次。第一层是硬件信任根TPM构成的可信虚拟平台底层，可以从物理硬件层次保证虚拟化平台的可信；第二层主要包括VMM以及管理域Dom 0的相关组件，我们把管理域记作Dom0，针对不同的VMM，Dom0的启动会有不同的方式。例如，Xen 的管理域启动相关组件包括运行中涉及的VBOIS、VOSLoader、VMOS等组件，这一层次可以作为TVP-QT的可信计算基。值得指出的是，与已有的TVP不同，我们只把Dom0 Kernel看成是可信基，这显然更为合理，因为Dom0实际上是整个虚拟化平台的管理域，含大量的应用程序，这些管理程序无法采用TCG链式度量，且也很容易受到攻击而改变[28-31]。第三层是我们重点设计的可信衔接点，可信衔接点位于Dom0，是Dom0的一组应用程序，包括vTPM实例的创建模块vTPM Builder、vTPM-VM映射组件vTPM-VM Binding以及VM的创建组件VM Builder，且作为vRT的一部分，在信任链上按照vTPM Builder→vTPM-VM Binding→VM Builder的顺序依次进行度量。可信衔接点可对TVP-QT的第一、第二层与第四、第五层进行有效衔接，保证TVP-QT信任链构建的连贯性，起到承上启下的作用，具有瀑布特征。第四层为vTPM，vTPM作为可信虚拟机部分的虚拟信任根，是由可信衔接点为虚拟机创建的vTPM实例，它可利用TCG的动态度量信任根（DRTM）机制启动，作为虚拟化平台应用进程的一部分。最上层为可信虚拟化平台上与用户活动联系最大的用户虚拟机层次，其运行时组件主要包括VBIOS、VOSLoader、VMOS、应用程序（APP）等相关组件。基于上述对TVP-QT的分析，本文从功能角度给出TVP-QT的抽象定义。

显然，相对于已有的TVP，我们提出的TVP-QT运行架构具有如下特点：

（1）TVP-QT更加精细。已有的TVP把整个管理域作为TCB，包括Dom0 Kernel和所有应用程序，而TVP-QT模型仅把Dom0 Kernel、TJP及vTPM作为TCB，由于TJP及vTPM只是管理域中很小一部分应用程序，因此TVP-QT的TCB更小。

（2）TVP-QT在逻辑上也更合理。一方面，已有的TVP的TCB无法采用TCG链式度量机制进行安全保证，而TVP-QT的TCB都能。因此，TVP-QT更符合TCG的链式度量标准。另一方面，TVP-QT增加了TJP，从逻辑上比已有的TVP更加合理。

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

**3.2 TVP-QT组件定义**

定义1 TVP-QT是具有可信功能的虚拟化计算平台，主要包括2类功能组件：TVP-QT:={M, RT}，M表示虚拟化平台所有主机类型集合，包括构成虚拟化平台的基本组件VMM、管理域内核、可信衔接点及用户虚拟机等，它们是利用虚拟化技术为用户提供资源与服务的主体；信任根（Root of Trust, RT）是构建 TVP信任环境的基础，也是 TVP 的核心组件，对虚拟化平台来说，它包括硬件TPM 、可信衔接点TJP和 vTPM。

对于 TVP 的主机 M，根据其类型进一步细化为 M:={m, vm}，其中，m:={VMM, Dom0 Kernel, TJP}，特指底层的VMM、Dom0 Kernel和可信衔接点TJP，它们是 TVP 的TCB。vm:={vm1, … , vmn}，表示虚拟化平台上层的用户虚拟机vm 集合。

相似地，TVP的信任根也进一步分类为RT:={TPM, vRT}={TPM, (TJP, vTPM)}，其中，TPM 是底层的硬件信任根，主要通过可信计算技术为物理平台提供信任保障，它拥有可信平台的非易失存储、密钥存储等固有特性；vRT包含可信衔接点TJP和vTPM，在功能实现上可表现为主机m中内核组件或部分独立的可信组件，本文将vRT抽象为一个独立的可信功能组件，通过特定的映射关系与硬件信任根TPM关联，以确保其vRT的可信性。其中vTPM是软件形式的TPM，具有TPM的安全功能；TJP 是可信衔接点， TJP的可信依赖于物理TPM，用来衔接底层的虚拟化平台m和顶层的虚拟机vm。

因此，TVP 从功能角度可定义为：

TVP:= {(TPM, m), (vRT1, vm1), … , (vRTn, vmm)}

= { (TPM, (vmm, Dom0 Kernel, TJP) ), ((TJP, vTPM1), vm1), …,( (TJP, vTPMn), vmn) }

其中，m必须使用 TPM 来构建信任，而虚拟机vm则是利用TJP和其相应的vTPM来构建信任。

特别地，可信衔接点TJP可以划分为TJP:= { vTPM Builder, vTPM-VM Binding, VM Builder}，其中vTPM Builder, vTPM-VM Binding, VM Builder都作为可信平台管理域上应用程序的一小部分。vTPM Builder表示与创建和管理vTPM实例的相关组件，并负责提供给vm运行时的vTPM标识以及端口；而vTPM-VM Binding则表示对vm和vTPM实例间绑定关系的相关组件，VM Builder表示与创建用户虚拟机相关的配置文件、组件等。在TVP-QT涉及到的vTPM架构中，每个vm必须与唯一对应的vTPM实例绑定。可信衔接点TJP的来源如表1所示，其中vTPM管理组件可由VMM上的vTPM组件提供，比如目前存在与Xen上的vTPM Manager。

TJP功能组件来源

|  |  |  |
| --- | --- | --- |
| TJP | 主要组件 | 主要功能组件来源 |
| vTPM Builder | vTPM启动组件 | vTPM管理组件 |
| vTPM配置文件 | 虚拟化平台 |
| vTPM-VM Binding | vTPM-VM绑定组件 | vTPM管理工具、  VMM |
| VM Builder | VM启动组件 | VMM |
| VM配置文件 | 虚拟化平台 |

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

**3.3 TVP-QT工作程序**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

**3.3 基于Xen的TVP-QT实现**

××××××××××××××××××××××××

**3.4 基于Xen的TVP-QT实验**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

3.4.1 **实验环境**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××我们

基于Xen实现了TVP-QT的原型系统，并进行仿真实现和结果分析，对TVP-QT信任链进行有效性验证和性能测试。下面对仿真实验的环境进行描述。

使用TPM-Emulator对TPM功能进行仿真模拟，实验的Xen VMM版本为Xen4.4.0[40][41]，实验物理平台的配置为Intel Core i3 @3.4GHz处理器，内存为8GB，物理存储为1T。Dom0采用Ubuntu LTS14.04，内核版本为Linux3.19.0，DomU使用类型为Ubuntu LTS14.04的半虚拟化虚拟机，内存为4GB，并且部署不同的应用作为仿真实验的测试对比。

下表为TVP-QT实验环境所用物理平台和DomU类型为Ubuntu的具体配置信息：

物理平台(Dom0)和用户虚拟机(DomU-Ubuntu)配置

|  |  |  |
| --- | --- | --- |
| 配置项 | 物理平台(Dom0特权域) | 用户虚拟机(DomU-Ubuntu) |
| CPU | Intel Core i3 @3.4GHz | Intel Core i3 @3.4GHz |
| 内核版本 | Linux3.19.0 | Linux3.19.0 |
| 内存 | 8G | 4G |
| 二级缓存 | 4M | 4M |
| 硬盘容量 | 1T | 30G |

以下图示表示在Dom0上创建DomU类型为Ubuntu LTS14.04的配置文件部分参数，以及实验所需vTPMManager域以及vTPM实例域的配置参数。

图9为类型为Ubuntu的用户虚拟机配置：

UbuntuTest1.cfg

Kernel = "/boot/vmlinuz-3.19.0-25-generic"

ramdisk = "/root/xen-image/UbuntuTest1.img"

name = "UbuntuTest1"

memory = "4096"

disk = [ 'file:/root/xen-image/UbuntuTest1.img,sda1,w' ]

vtpm=["backend=vtpm-UbuntuTest1"]

.……

DomU-Ubuntu配置部分参数

The Configure of DomU-Ubuntu

图10为UbuntuTest1对应的vTPM实例配置文件：

vtpm-UbuntuTest1.cfg

name="vtpm-UbuntuTest1"

Kernel="/usr/lib/xen/boot/vtpm-stubdom.gz"

extra="loglevel=debug"

memory=8

disk=["file://root/xen-images/vtpm-UbuntuTest1.img,sda1,w"]

vtpm=["backend=vtpmmgr,uuid=ac0a5b9e-cbe2-4c07-b43b-1d69e46fb839"]

.……

Ubuntu vTPM实例配置部分参数

The Configure of Ubuntu vTPM instance

图11为vTPMManager域配置文件：

vtpmmgr.cfg

name="vtpmmgr"

Kernel="/usr/lib/xen/boot/vtpmmgr-stubdom.gz"

extra="tpmlocality=2"

memory=8

disk=["file:file://root/xen-images/vtpmmgr-stubdom.img,hda,w"]

iomem=["fed42,1"]

……

vTPMManager配置文件部分参数

The Configure of Ubuntu vTPMManager

3.4.2 **功能实验**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

（1）

3.4.3 **性能实验**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

**3.5 本章小结**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

**4** 具有瀑布特征的TVP-QT信任链模型

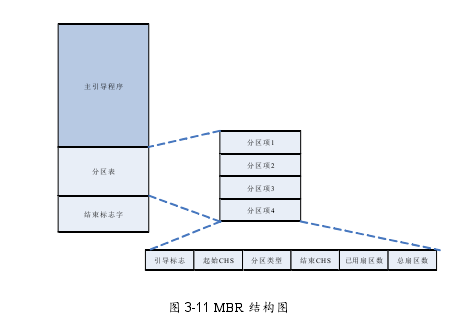
**4.1 TVP-QT信任链分析**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

4.1.1 **Linux启动分析**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××× 要进行操作系统启动的完整性度量，首先需要清楚启动流程。对任何一个操作系统而言，总体上来说，其启动过程主要分为两个阶段:硬件平台启动阶段和操作系统引导阶段。其中，硬件平台启动阶段包含系统平台加电、BIOS(基本输入输出系统)自检、对平台硬件的度量和启动、BIOS将控制权转交给Boot(通常是主引导扇区MBR，也可能是操作系统加载器，如Grub或者Lilo)等几个步骤。操作系统的引导阶段则包括Boot分析操作系统的引导扇区并移交控制权、操作系统引导扇区执行初始化工作并调入操作系统加载器OSLoader, OS Loader加载操作系统内核等几个步骤。

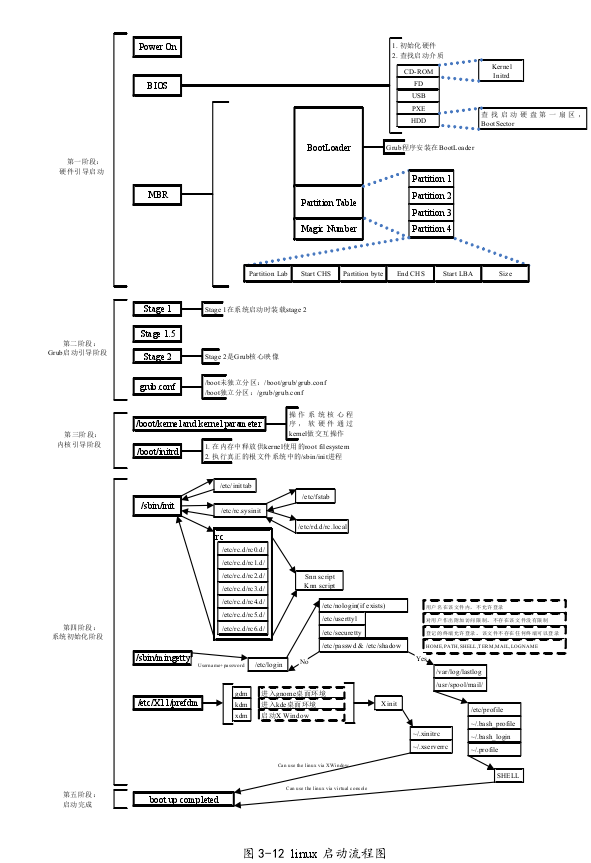
可信计算平台上Linux的启动步骤如下: ①开机初始化BIOS。在计算机中CS段寄存器会被CPU设为FFFFH，并且还会对其它的寄存器做出清零的操作，随后执行CS : IP处的指令。该处的指令实际上是BIOS的入口指令。之后BIOS进行自检，并对平台其他硬件设备进行启动和初始化的工作。 ② BIOS搜寻磁盘的主引导扇区MBR (Master Boot Record)，位于硬盘的0柱面、0磁头、1扇区，并把 MBR的BootLoade:调至内存的CS:IP=OOOOH:7COOH处。MBR共512字节，分为两个部分:第一部分是Pre-Boot部分，占446字节;第二部分是磁盘分区表信息，占64字节。最后两个字节是硬盘有效标志(SSAA)的标志。如图3-11所示。 ③BIOS把控制权转交给MBR o MBR将分析内部的磁盘分区表(Disk Partition TableDPT)，根据分区表信息来分析需要被度量的操作系统位于哪个磁盘的分区，并在这个分区的引导扇区的哪个地址，随后将控制权转交给引导扇区。



④引导扇区首先需要执行初始化工作，然后转交控制权给操作系统加载器OSLoader，并加载操作系统内核直到全部加载完成操作系统的核心内容。 在实际的应用中如果用户选择了某一操作系统并进行启动，Grub首先找到该操作系统的引导扇区并交出控制权。以Linux为例，其中bootsect. S文件是一种实模式下运行的汇编程序，在汇编后自动生成一组二进制的代码，并将其存储在操作系统的引导扇区中。Grub向bootsect. S移交控制权，并加载核心系统和setup. S o setup. S检查己调入内存的内核代码是否正确。如果检测的结果显示代码正确，其可以利用BIOS中断来实现对内存信息、设置显示器基本模式和硬盘信息的读取，然后，准备让CPU进入保护模式。

再经过一系列复杂的操作，当Linux核心加载完成后，它将对计算机的硬件和相关设备驱动执行初始化操作，随后将会在后台运行init进程。在系统的运行过程中第一个进程就是init进程，该进程是计算机系统中其它进程的起点，所有其他进程都是由init逐步调用“fork”系统得到的。Init在运行时将使用到//etc/initta b和//etc/rc.d这两个重要的引导配置文件。Inittab是对系统不同启动进程的一种描述，它能够将初始化中系统内部因为不同级别而分别进行的启动进行描述。在系统中整个启动级别有7级，其中最低一级是0级，最高的是6级，并与//etc/rc.d下的目录一一对应。

之后，由mingetty进行login操作。最后，当用户成功登录，在用户界面出现shell的时候，用户就可以使用操作系统了。linux启动流程如图3-12所示。



4.1.2 **Linux启动可信度量因素及内容分析**

在研究了Linux启动流程之后，还需要分析对Linux启动流程进行完整性度量时二虑的因素。主要为Linux启动过程的完整性。

单个组件的完整性:其目的主要是防止组件被破坏、感染病毒、植入木马等。

过程所需组件的完整性:在启动过程中所需要的组件不能缺失，而且不能被伪造，替换。典型的如LSM(Linux Security Module)模块。该模块是加强系统的安全性，在;统中如果 LSM遭受到了攻击，系统将失去对敏感数据的保护功能，而且很可能在操j系统内核引入恶意功能。又例如:rc. d系列配置文件的完整性。启动顺序的完整性:保证各个组件在启动过程中的顺序。单一过程中组件的启动j程中必须遵循一定的加载顺序，如果加载顺序出现错乱，有可能会导致系统处于不安二的状态，造成相关安全漏洞的出现。启动顺序的完整性实际上是由前面两点决定的，就是在单个组件完整，并且其所需组件也是完整的情况，启动顺序完整性是能够得到保证的。

分析了Linux启动过程和对启动过程进行完整性度量需要考虑的因素的基础上，就可以确定Linux启动流程需要进行完整性度量的内容了。

1. 主引导扇区MBR。当BIOS自检完毕，在检aJ}}和重置平台的其他硬件设备之后，将会把控制权转交给MBR。在MBR中有预启动代码和磁盘分区表等一系列重要信息，所以需要对其加以保护。
2. Grub引导系统。Grub通常分为以下三个阶段。

stagel: stagel会被直接的写入MBR中，这样计算机设备在启动时一旦对硬件的安全性检测完毕以后就会将系统的控制权转到GRUB中。stage 1随后则会被BIOS加到Ox7c00处并在这一处执行跳转操作。Stage 1 C /stage 1 /start. S)所需完成的任务很简单，仅仅只需要将0头磁盘的0道中的第二扇区的内容读到内存中即可。而0头0道2扇区内容是/stage2/start. S编译后512字节，它是stage2或者stagel-5的入口。在这一过程中stagel不具备对文件系统的识别功能。

stage2也是必须具有的，通常它存放在某个文件系统中。stage2是Grub的核心，基本上所有的Grub功能都是由stage2实现的。

stagel-5是可选的，它介于stage 1和stage2之间，当stagel不能够识别stage2所在的文件系统的时候，就会调用stagel-5来识别文件系统以便加载stage2 0。因此，Grub的实际运行流程是stager stagel-5 (optional)一stage2。这三部分都要度量。

(3)操作系统内核。内核的引导流程至少应包括两个内核启动的关键文件:initrd和vmlinuz。

①initrd 。initrd是“initial ram disk”的简称。initrd是Linux在系统引导过程中使用的一个临时根文件系统，用来临时把硬件引导到实际内核vmilnuz能够接管并继续引导。的状态。本质上initrd就是一个带有根文件系统的虚拟RAM盘，里面包含了根目录“/”以及其他目录，比如bin, dev, proc, sbin, sys等Linux启动时必需的目录。通常，Linux内核挂载真正的根文件系统是使用initrd来进行挂载的，挂载完成后把initrd从内存中卸载掉，因此，initrd是一个过渡性质的处理过程。

②vmlinuz o vmlinuz是一个压缩的可执行Linux内核文件。所谓的内核，是指计算机操作系统最核心的程序，它在启动过程中被装入物理内存中，并且在整个计算机运行的过程中都驻留在内存里。所谓的可执行，是指vmlinuz可以像一个程序一样能够被计算机所运行。vmlinuz文件是Linux内核加载的核心文件。

Vmlinuz首先被misc\_c中定义的函数decompresses kernel{)解压到地址Ox10flflOfl。然后，当CPU跳转到Ox1fl时，将执行“archili3861 kernelllead\_ S”中的startup 3 2 0startup一2是vmlinuz的入口，经过startup\_ 32，将会跳转到start\_ kernel()函数。start一ernel()在“initlmain. c”中被定义。Start一ernel{)调用一系列初始化函数，建立Linux核心环境。当startes kernel()执行完毕，基本的Linux核心环境就建立完成了。

（4）所有进程的起点init。它是系统中其它进程的起点，进程号是1 o Linux执行init程序是在完成内核引导之后进行的。对于这一部分，除了要保证init文件本身的完整性以外，还要保证init所读取的一些配置文件(或脚本)的完整性，这包括//etc/ inittab ,/etc/rc. d/rc. sysinit和/etc/rc.d/rc等。

(5) mingetty。当init重新从rc返回获取控制权的时候，它将启动mingetty o mingetty包括如下功能:打开终端，设置模式;输出登录界面和提示，接收用户名的输入;以用户名为login参数，加载login程序。 与此同时，与mingetty和login相关的登录配置文件的完整性也要得到保证，这包括/etc/nologin, /etc/securetty, /etc/passwd等。

(6)与启动流程相关的其他文件。当用户成功登录，出现shell的时候，Linux的启动就可以认为结束，用户就可以使用操作系统了。在Linux启动的整个过程中，可能还

有其他一些与启动流程相关的文件，例如用户使用的shell文件、内核符号表/boot/System.map和一些启动配置文件等，这是由启动流程、系统配置和用户自己定义的。

(7)用户自定义文件。对用户自定义文件的完整性度量，例如一些内核模块等。内核模块在操作系统内核启动完毕之后将会被加载到操作系统内核，成为整个内核的一部分。因此，对于内核模块的加载除了要施加访问控制以外，也需要保证其完整性。否则，一个被恶意篡改的内核模块一旦被加载，将可能导致整个操作系统进入不可信状态。虽然这些内核模块并不在启动流程之中，但是由于其高敏感性，其完整性也必须得到保证。

×××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

4.1.3 **Linux虚拟机启动可信度量因素及内容分析**

×××××××××××

4.1.4 **TVP-QT信任链分析**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

**4.2 TVP-QT信任链设计**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

**4.3 基于Xen的TVP-QT信任链构建**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××为了在实际系统中检验TVP-QT及其信任链，本文选择已经构建的实例系统进行分析。该实例系统基于 Xen半虚拟化平台，如图8；



图8基于Xen的TVP-QT系统

其中，vRT被分散在Dom0、vTPMmanager域和vTPM域。本节我们根据第3节中对TVP-QT信任链的描述：第一层硬件TPM（CRTM）→第二层TCB（BIOS→OSLoader→VMM→Dom0 Kernel）→第三层可信衔接点（vTPM Builder→vTPM-VM Binding→VM Builder）→第四层vTPM（vTPM实例）→第五层可信虚拟机（VBIOS→VOSLoader→VMOS→APP），将TVP-QT信任链分为三部分，第一部分就是虚拟化平台，包括TVP-QT信任链的第一层和第二层，第二部分是可信衔接点TJP，就是TVP-QT信任链的第三层，第三部分是用户虚拟机，就是TVP-QT信任链的第四层和第五层。接下来我们结合Xen 4.4系统，对这三部分信任链进行实际的分析与讨论。

对于第一部分，在Xen平台硬件加电启动之后，把CRTM作为整个信任链的起点，并由CRTM首先度量物理平台BIOS和其他有关BIOS的配置，然后BIOS获得系统的控制权并度量Xen的引导程序Grub，主要度量grub-xen(‍head.S, trampoline.S, x86\_32.S)，Grub获得控制权后会根据Xen的镜像头信息获得入口地址Oxl0000后读入Xen的镜像，并对此镜像和\_\_startxen()并进行度量，然后把控制权交给Xen，Xen获得信任之后对Dom0相关组件进行度量，包括construct\_dom0()、\_start\_32\_、start\_Kernel和LinuxOS镜像等。然后把控制权交给Dom0。至此，第一部分可信引导结束。

对于第二部分，Dom0 Kernel获得控制权后首先度量TJP的vTPM Builder，包括Xen中创建vTPMManager域的配置文件（.cfg）、vTPM Manger域（主要是MiniOS镜像文件和vTPM Manger程序）以及启动vTPM的vtpm-common.sh、vtpm-impl.sh等组件。然后把控制权交给vTPM Builder，vTPM Builder获得控制权，对TJP的vTPM-VM Binding进行度量，包括Xen中xl、xenstore、vtpmd、tpm-xen、vtpm\_manager\_handle等针对vTPM-VM绑定的组件。随后vTPMBuilder把控制权交给vTPM-VM Binding，vTPM-VM Binding获得控制权后，对TJP的最后的组件VM Builder进行度量，包括Xen的xl、libxl（Xen4.1之后xl作为默认的管理工具）等创建虚拟机所需的组件以及创建虚拟机的配置文件（.cfg）和虚拟机的镜像文件（.img）。完成VM Builder可信度量后，VM Builder获得信任链控制权。至此，第二部分可信信任链传递结束。

对于第三部分，完成度量VM Builder后，可以采用两种方法对vTPM进行度量，其一是静态度量，其二是动态度量。如果采用静态度量，控制权在VM Builder，如果采用动态度量，则控制权在物理TPM。但无论是静态度量还是动态度量，度量的对象都是vTPM实例域，包括vTPM实例域的配置文件(.cfg)以及启动文件(.img)和Mini OS、tpm instance等组件。由于目前较新的DRTM机制对其保护的应用有诸多限制，比如要求受保护的代码自包含等，因此我们采用对vTPM实例域采用静态度量方式。VM Builder完成对vTPM实例域的度量后，把控制权交给vTPM实例域，vTPM实例域获得控制权，对最后的DomU部分进行可信度量，包括DomU启动的内核所需启动信息页的有关的xen.h、start\_info、qemu-dm、qemu-xen、pc-bios等组件和linux镜像文件进行度量。需要说明的是，Xen中虚拟机有关的BIOS、引导等组件是利用封装在Xen中的Qemu实现的，所以需要对Xen中Qemu重要组件进行可信度量，如qemu-io、qemu-img等。在DomU启动相关组件完成度量之后，可信虚拟平台最后一部分信任链完成可信度量和信任传递。

以上述实例系统为例，我们完整展示了本文建立的通用抽象模型。值得注意的是，本实例系统的信任链得以正确传递需要满足以下前提：

（1）必须保障vRT:=TJP+vTPM自身的可信。在实例系统中，可信衔接点TJP包含的组件比较多，不仅大量应用程序、支持库和大量配置文件，而且还涉及Dom0、vTPM mana- ger和vTPM等域，需要度量的内容多，不允许出现遗漏，特别是TJP和vTPM关键的组件和配置文件必须是被度量的对象。

（2）必须确保TJP中的vTPM Builder、vTPM-VM Binding、VM Builder三个管理程序在启动时按顺序执行。尽管vTPM Builder、vTPM-VM Binding和VM Builder是Dom0中的应用程序，但必须保证按顺序执行才能度量结果。

**4.4 基于Xen的TVP-QT信任链实验**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

4.4.1 **实验环境**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

4.4.2 **功能实验**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××TVP-QT信任链在此虚拟化平台进行有效性测试时，利用哈希函数对信任链各层次的构建模块、功能组件或文件进行哈希值存储。按照TCG标准，采用迭代计算Hash值的方法对PCR进行扩展操作，将PCR的现值与新值相连，计算Hash值作为新的完整性度量值存储到PCR中，描述如下：

New PCRi=Hash(Old PCRi||New Value)，

其中，Hash 函数选用SHA-1，||表示连接符号。在实验中成功运行虚拟机UbuntuTest1。按照下表的顺序对PCR进行存储。其中PCR[0]-[7]存储TVP-QT信任链第一层到第二层TCB的可信度量信息；PCR[8]-PCR[10]分别存储信任链中可信衔接点三个重要的组件的度量信息；PCR[11]-PCR[15]存储vTPM实例域和用户虚拟机信任链度量信息。具体的存储如表3所示：

表3仿真实验PCR存储简述

|  |  |  |
| --- | --- | --- |
| **寄存器** | **存储内容** | **功能层次** |
| PCR[0]-  PCR[7] | BIOS代码 | 第一层：  硬件TPM  第二层：  TCB |
| 可信云平台配置信息 |
| Xen引导xen-grub  (‍head.S,trampoline.S,x86\_32.S等) |
| Xen VMM内核代码(Xen Kernel) |
| Dom0OS启动相关信息  (construct\_dom0()､\_start\_32\_､start\_kerne等) |
| Dom0 OS Kernel |
| PCR[8] | MiniOs 及vTPMManager配置文件(vTPM Builder，.cfg、.img以及vtpm-common.sh、vtpm-impl.sh等组件) | 第三层：  可信衔接点 |
| PCR[9] | 负责vTPM-VM相关组件  (xl、xenstore、vtpmd、tpm-xen、vtpm\_manager\_handle等组件) |
| PCR[10] | VM配置文件(VM Builder，xl、libxl以及.cfg、.img等组件) |
| PCR[11] | MiniOs 及vTPM实例域（.cfg,.img,tpm instance等组件） | 第四层：  vTPM |
| PCR[12] | VBIOS及其他虚拟BIOS配置信息(qemu-dm、qemu-xen、pc-bios等组件) | 第五层：  可信虚拟机部分 |
| PCR[13] | VOSLoader(虚拟机启动引导文件)，如Linux 系统的initrd和vmlinuz 文件。 |
| PCR[14] | VM启动的其他信息  (xen.h、start\_info、qemu-io、qemu-img等组件) |
| PCR[15] | VM中的应用程序 |

按照TVP-QT信任链顺序存储的信任链信息结果如图12 所示。只要程序或者文件不发生任何变化，即使反复执行或者查看，信任链中不会重复记录程序或者文件的哈希值，哈希值也不会发生变化｡而一旦程序或者文件内容发生变化，下次执行该程序或者打开该文件时，就不可避免的在信任链中留下痕迹，用户虚拟机使用者就可以据此判断平台状态是否可信｡

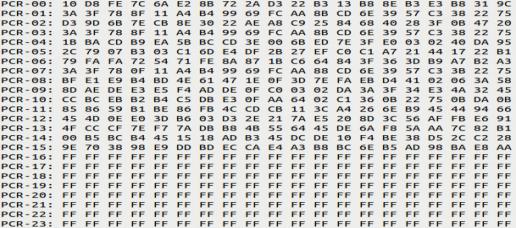


图12 信任链PCR信息

4.4.3 **性能实验**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××与已有的TVP信任链模型相比，TVP-QT信任链模型增加了可信衔接点TJP。TJP包含的组件vTPM Builder、vTPM-VM Builder 和VM Builder无论作为Dom0的内核组件还是作为Dom0的应用，均需要再独立度量，因此，无论是对底层m信任链的构建还是顶层vm信任链的构建均会带来额外的开销。对于底层m信任链的构建，TVP-QT信任链模型比已有的TVP信任链模型增加了对TJP的静态度量；对于顶层vm信任链的构建, TVP-QT信任链模型比已有的TVP信任链模型增加了对TJP的动态度量。

为此我们首先针对TVP-QT信任链构建过程中有关主机m的信任链构建进行性能测试和结果分析，并与传统的TVP架构（图1所示）进行对比；然后针对TVP-QT信任链构建过程中有关vm的信任链构建进行性能测试和结果分析。值得注意的是，与传统的TVP信任链相比，vm信任链的构建过程仅仅多了对TJP的动态度量。

本节性能测试的实验环境采用表1所描述的物理平台Dom0(Ubuntu LTS 14.04)和用户虚拟机DomU(Ubuntu LTS 14.04)，并且在Dom0和DomU分别安装一些常用软件来模拟云计算开发环境和云用户环境，比如Firefox[41]、WPS for Linux[42]、Wine[43]、Eclipse[44]等。下面本文分别针对在TVP-QT和传统TVP下m、vm的信任链构建实验，对性能方面进行对比和分析。

4.4.3.1 信任链构建的性能分析

传统TVP信任链中主机m的信任链构建过程为：

CRTM→BIOS→OSLoader→VMM→Dom0 OS Kernel →app；

TVP-QT中主机m的信任链构建过程：

CRTM→BIOS→OSLoader→VMM→Dom0 OS Kernel →vTPM Builder →vTPM-VM Binding→VM Builder→other\_app

我们对以上两条信任链进行10次实验，并记录每次的完成时间。如图13所示。



图13 m信任链构建时间

由图13可知，虽然TVP-QT在主机m上比传统TVP多了TJP的静态度量，但是在时间上并没有太大的多余开销，对可信虚拟的运行的影响不大。所以，TJP的引入可以在保证可信虚拟平台m相关组件实现完整度量的情况下，不会给平台带来太多的开销。

4.4.3.2 vm信任链构建的性能分析

传统TVP信任链中vm的信任链构建过程为:

INIT→VBIOS→VOSLoader→VMOS→APP

TVP-QT中主机vm的信任链构建过程：

(TJP)TPM\_Dynamic→vTPM→VBIOS→VOSLoader→VMOS

→APP

我们对以上两条信任链进行10次，并记录每次的完成时间。如图14所示。



图14 m信任链构建时间

由图14可知，TVP-QT相比传统TVP下对vm的信任链构建过程，也仅仅多了由TJP带来的额外开销，可以保证对vm可信度量后的正常启动。

综上对TVP-QT与TVP信任链构建过程的对比实验来看，带有可信衔接点TJP的可信虚拟平台TVP-QT能够保证在对整个平台带来足够小的开销的情况下实现对平台的可信度量，保证了虚拟化环境的安全可信。

**4.5 本章小结**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××信任链TVP-QT信任链的构建过程进行了详细的描述，通过仿真实验对TVP-QT及其信任链的有效性和性能等进行了测试，证明了该信任链的正确性和有效性。

**5** TVP-QT信任链形式化分析

**5.1 基于扩展安全逻辑(LS2)的TVP-QT分析**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

5.1.1 **基本假定**

×××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××在对TVP-QT信任链属性进行形式化分析前，本文假定以下条件是成立的：

（1）TVP 中各个层次的系统镜像文件（包括主机m以及用户虚拟机层次上的各个用户虚拟机VM）的完整性未受破坏，并且各个用户虚拟机都预先植入所需要的可信度量和证明代理功能组件；（2）主机m支持动态加载动态度量根 DRTM技术，能够为TJP和vTPM提供动态的可信运行环境；（3）vTPM的平台身份密钥（Attestation Identity Key，AIK）已得到可信第三方的认证并颁发证书，这里不考虑其具体实现方案（参见vTPM[14]及 Trust Visor[38]等）；（4）远程验证方案基于 TCG 组织给出的完整性报告协议，且在远程挑战者 R 与本地 TVP 之间已经建立了安全信道[19]。

从2.2的分析可知，本文对TVP-QT 信任链的信任属性分析验证主要包括3 部分：

（1）m信任链构建的验证及该信任链的远程验证（含TJP）；

（2）TJP动态度量验证及远程验证；

（3）利用vTPM构建的vm信任链验证及远程证明；

5.1.2 **主机m信任链的本地验证及远程证明**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

5.1.2.1 本地程序执行

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

根据2.2节对TVP中m信任属性TPm定义以及TPvRT中对TCTJP的定义，其信任链本地执行过程中涉及到的程序如图4所示。

*SRTM*(*m*) ≡ *b* = read *m.bios\_loc*

Extend *m.pcr.s*,*b*;

Jump *b*

*BIOS*(*m*) ≡ *o* = read *m.os\_loader\_loc*

Extend *m.pcr.s*,*o*;

Jump *o*

*OSLoader*(*m*) ≡ *v* = read *m.vmm\_loc*

Extend *m.pcr.s*,*v*;

Jump *v*

*VMM*(*m*) ≡ *d* = read *m.dom0\_Kernel\_loc*

Extend *m.pcr.s*,*d*;

Jump *d*

*Dom0 Kernel*(*m*)≡ *vb*= read *m.tjp\_loc*

Extend *t.pcr.s*,*t*;

Jump *vb*

*vTPM-Builder*(*m*) ≡ *vv*= read *m.vtpm-vm-binding\_loc*

Extend *m.pcr.s*,*vv*;

Jump *vv*

*vTPM-VM Binding*(*m*) ≡ *vmb* = read *m.vm-builder\_loc*

Extend *m.pcr.s*,*vm*;

Jump *vmb*

*VM-Builder*(*m*) ≡ o\_*app* = read *m.* o\_*app \_loc*

Extend *m.pcr.s*, o\_*app*;

Jump o\_*app*

*Other\_APP*(*m*) ≡ ...

TVP-QT中 m 信任链传递

程序执行流程：m首先从CRTM启动执行，它从主机内存地址*m.bios\_loc*中读取BIOS的代码*b*，将其扩展到一个PCR中（其中，*m.pcr.s*表示该主机在这里存储所有相关度量值，且该主机的度量值存储于静态度量的PCR中）,之后执行指令Jump *b*；然后CRTM将控制权传递给m的BIOS，它从主机内存地址*m.os\_loader\_loc* 中读取的OS\_Loader代码*o*，将其扩展到一个PCR中,之后执行指令Jump *o，*将控制权交给OSLoader；OSLoader继续按序从内存*m.vmm\_loc*读取VMM的代码*v*，将其扩展到*m.pcr.s*，然后转换控制权给VMM，VMM、Dom0 Kernel执行相似流程，直到可信衔接点TJP的加载。

5.1.2.2 本地可信属性描述

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××由上文描述的信任链传递所涉及的程序执行过程可知，体现主机m信任链的是主机进行可信度量后的PCR值，它与执行程序之间存在着唯一确定的映射关系。因此，基于定义2及上述映射关系，可将m的本地信任传递属性归纳为：如果可信度量后的PCR中度量值序列是正确的值，那么在该虚拟机上信任链所加载的程序顺序就是正确的。即m的本地信任传递属性就是要求所有相应启动程序如BIOS、OSLoader、VMM、Dom0 Kernel、vTPM Builder 、 vTPM-VM Binding、VM Builder等都能按确定的先后顺序加载。以LS2将这种顺序形式化表示为

MeasuredBootSRTM(m,*t*)=



 (Reset(*m*,*J*)@*tS*)(Jump(*J*,*BIOS*(*m*)) @*tb*)

(Jump(*J*,*OSLoader*(*m*))@*to*)(Jump(*J*,*VMM*(*m*))@*tv*)

(Jump(*J*,*Dom0\_Kernel*(*m*))@*td*)

(Jump(*J*, *vTPM-Builder*(*m*) )@ *tvb*)

(Jump(*J*, *vTPM-VM Binding*(*m*) )@ *tvv*)

(Jump(*J*, *VM-Builder*(*m*) )@ *tvvb*)

(Jump(*J*, *VM-Builder*(*m*) )@ *to\_app*)

(┐Reset(*m*)on(*tS*,*t*])  (┐Jump(*J*)on(*tS*,*tb*))

(┐Jump(*J*)on(*tB*,*to*)) (┐Jump(*J*)on(*to*,*tv*))

(┐Jump(*J*)on(*tv*,*td*))(┐Jump(*J*)on(*td*, *tvb*))

(┐Jump(*J*)on(*tvb*, *tvv*))(┐Jump(*J*)on(*tvv*, *tvvb*))

(┐Jump(*J*)on(*tvvb*, *to\_app*))

上述公式表示：如果TVP的m基于信任链构建了本地信任环境，则其启动过程一定是从BIOS跳转到OSLoader，从OSLoader到VMM，从VMM到Dom0\_Kernel，然后Dom0\_ Kernel到TJP，而在此期间不会有其他程序执行。这就需要证明上述程序启动序列与PCR值之间的一一映射关系。基于前文的假定前提，要证明的信任链本地信任属性如下。

**定理1** 如果m从CRTM启动运行，且与该m启动过程对应的PCR值为

*seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),*Dom0\_Kernel*(*m*), *vTPM Builder*(*m*), *vTPM-VM Binding*(*m*), *VM Builder*(*m*))，

那么该m的本地信任链传递过程就是唯一的、正确的，即确定地从*BIOS*(m)到*OSLoader(*m)再*VMM*(m)、*Dom0 Kernel*(m)、*vTPM Builder*(m) 、 *vTPM-VM Binding*(m)、*VM Builde*(m)*r*。该信任属性形式化表示为

ProtectedSRTM(m)+

Mem(*m.pcr.s*,*seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),*Dom0\_Kernel*(*m*),*vTPMBuilder*(*m*), *vTPM-VM Binding*(*m*), *VM Builder*(*m*))MeasuredBootSRTM(m,*t*)

**证明：** 本文按照以下步骤进行证明：

首先，由前提条件可知在时间点*t*，有

Mem(*m.pcr.s*,*seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),*Dom0\_Kernel*(*m*),*vTPM Builder*(*m*), *vTPM-VM Binding*(*m*), *VM Builder*(*m*))

成立，反复利用PCR公理即可直接得到在该序列中的所有子序列一定在时间*t*之前就出现在*m.pcr.s*中，即：

*tS*,*t1*,*t2*,*t3*,*t4*, *t5*,*t6*,*J*.(*tSt1*<*t2*<*t3*<*t4 <t5* <*t6*<*t*)

(Mem(*m.pcr.s*, *seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),

*Dom0\_Kernel*(*m*),*vTPM Builder*(*m*),

*vTPM-VM Binding*(*m*), *VM Builder*(*m*))*@t*)

(Mem(*m.pcr.s*, *seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),

*Dom0\_Kernel*(*m*), *vTPM Builder*(*m*), *vTPM-VM Binding*(*m*) *@t6*)

(Mem(*m.pcr.s*, *seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),

*Dom0\_Kernel*(*m*), *vTPM Builder*(*m*), *@t5*) (Mem(*m.pcr.s*, *seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),

*Dom0\_Kernel*(*m*))*@t4*)

(Mem(*m.pcr.s*,*seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*)))@*t3*)

(Mem(*m.pcr.s*,*seq*(*BIOS*(*m*),*OSLoader*(*m*))@*t2*)

(Mem(*m.pcr.s*,*seq*(*BIOS*(*m*))@*t1*)Reset(m,*J*)@*tS*

(┐Reset(*m*)on(*tS*,*t*) (**1**)

接下对图4中信任链的执行过程进行说明，最先执行的操作是以CRTM为起点启动m，即Reset(m,*J*)，然后m执行第一个信任程序*BIOS*(m)。利用LS2规则，在某个时间*t*B，程序会跳转到*b*,且其他时间不会有程序跳转，内存位置（即PCR值）被该线程锁定，即有以下属性(2)成立。



(((Mem(*m.pcr.s*,*seq*(*BIOS*,*b*,*o*))@*t’*) (< *t’< t*))

((< *< t*) (Jump(*J*,*b*)@ )))

(IsLocked(*m.pcr.s*,*J*) @ ) **(2)**

类似地，接下来的信任程序：

*OSLoader*(*m*),*VMM*(*m*),*Dom0\_Kernel*(*m*),*vTPM Builder*(*m*), *vTPM-VM Binding*(*m*), *VM Builder*(*m*)，

也利用LS2规则，在某个时间*to*、*tv*、*td*、*tvb*、*tvv*、*tvmb*、*to\_app*，程序会跳转到*o、v、d、vb、vv、vmb、o\_app、*,且其他时间不会有程序跳转，相应的内存位置（即PCR值）被该线程锁定，即有属性(3)、（4）、（5）、（6）、（7）、（8）、（9）成立，鉴于篇幅，这些属性略。

根据式（2）—（9）可知，如果前提条件满足，那么*m*上执行程序的顺序一定是从BIOS(m)到OSLoader(m)再到VMM(m)、Dom0 Kernel(m)、TJP(m)。





 (┐Reset(*m*,*I*)on (*tS*,*t*])  Reset(*m*,*J*)@ *tS*)

(Jump(*J*,*BIOS*(*m*)) @*tb*) (┐Jump(*J*,*BIOS*(*m*)) on(*tS*, *tb*))

(Jump(*J*,*OSLoader*(*m*))@*to*)

(┐Jump(*J*,*OSLoader*(*m*)) on(*tb*,*to*))

(Jump(*J*,*VMM*(*m*))@*tv*)

 (┐Jump(*J*,*VMM*(*m*)) on(*to*, *tv*))

(Jump(*J*,*Dom0\_Kernel*(*m*))@*td*)

 (┐Jump(*J*,*Dom0\_Kernel*(*m*)) on(*tv*, *td*))

(Jump(*J*, *vTPM-Builder*(*m*) @*tvb* )

 (┐Jump(*J*, *vTPM-Builder*(*m*) on(*td*, *tvb*))

(Jump(*J*, *vTPM-VM Binding*(*m*))@ *tvv*)

 (┐Jump(*J*, *vTPM-Builder*(*m*) on(*tvb*, *tvv*))

(Jump(*J*, *VM Binding*(*m*))@ *tvmb*)

(┐Jump(*J*, *VM Builder*(*m*) on(*tvv*, *tvmb*))

(Jump(*J*, *VM Binding*(*m*))@ *to\_app*)

(┐Jump(*J*, *VM Builder*(*m*) on(*tvmb*, *to\_app*)) **(10)**

定理1即得证。

虽然上述证明过程未显式地描述攻击者的存在，但已经蕴含着攻击场景。比如，在*BIOS*(*m*)之后跳转到*o*的过程中，由于*o*是从内存*m.osloader\_loc*读取的，而该位置可能在之前已被攻击者线程写入其他程序，但可信计算技术提供的度量扩展机制使得能够推理只有得到正确的内存值时才能继续运行下一个程序。后面的以此类推。

5.1.2.3 信任链远程验证

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

TVP-QT的m需要向外部挑战者证明自己所声称信任属性，即其信任链传递过程中所执行程序的确定序列，使外部挑战者相信它的确按上述信任链构建了可信执行环境，需要证明MeasuredBootSRTM(m,t)成立。

**a.远程验证程序执行**

首先，根据 TCG 远程证明协议规范及在虚拟化平台中的实现，给出m 信任传递的远程验证过程中涉及到的程序，如图5 所示。

*TPMSRTM*(*m*) ≡ *w =* read *m.pcr.s*;

*r =* sign(*PCR*(*s*),*w*),*AIK-1*(m);

send *r*

*Verifier*(*m*)≡ *sig =* recieve;

*v =* verify *sig*, *AIK*(*m*);

match *v* ,(*PCR*(*s*);

*seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),*Dom0\_Kernel*(*m*),*vTPM Builder*(*m*),*vTPM-VM Binding*(*m*), *VM Builder*(*m*))

1. TVP-QT中m信任传递的远程验证程序

首先，m读取本地存储的PCR值，用自己的AIK签名（*AIK-1*(*m*)）并将其发送给挑战者。然后，挑战者验证该签名，并用预期的度量值序列与收到的值进行对比，如果PCR值是匹配的，则表明该m拥有所声称的可信属性，否则验证失败。在此过程中远程验证者与m应是不同实体，以保证该验证过程的有效性。

这些前提条件形式化表示为

SRTM = {*AIK*(*m*),Honest(*AIK*(*m*),{ TPM SRTM(*m*),TPM DRTM(*m*)} ) } (11)

**b.信任链属性的远程验证**

根据远程证明协议执行流程，给出以下信任传递属性的远程证明目标。

**定理 2**如果远程验证者确认m提供的度量值是唯一的、正确的，那么该m对应的PCR值一定是如下的确定序列*seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),*Dom0\_Kernel*(*m*),*vTPM Builder*(*m*), *vTPM-VM Binding*(*m*), *VM Builder*(*m*))，因为根据定理1可知，该序列表明m的确执行了相应的信任链传递过程。

形式化表示为：

SRTM├ [Verifier(*m*)]*t.*(*t<te*)

(Mem(*m.pcr.s*, *seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),

*Dom0\_Kernel*(*m*),*vTPM Builder*(*m*), *vTPM-VM Binding*(*m*),

*VM Builder*(*m*))@*t*) (**12**)

SRTM，ProtectedSRTM(*m*)

├ [Verifier(*m*)]*t.*(*t<te*)

MeasureBootSRTM(*m*,*t*) (**13**)

这两个属性有递进关系，即如果属性(12)成立，则属性(13)可以利用定理1的结论直接证明。因此，下面对属性(12)进行证明。

**证明：**首先根据前提假设及[Verifier(*m*)]，利用公理*VER*可得到：

[Verifier(*m*)]*tf*,*e*,*I.*(*tf<te*)

= (*m*)Contain(*e*,*SIGAIK*(*m*)*-1*)

{|*PCR*(*s*), *seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),

*Dom0\_Kernel*(*m*),*vTPM Builder*(*m*), *vTPM-VM Binding*(*m*), *VM Builder*(*m*)) )|})(Sent(*I*,*e*)@ *tf*)

*l.*(Write(*I*,*l*,*e*)@ *tf*))

根据图5中的远程验证程序，建立并证明以下程序不变量：对于程序前缀*QIS*(CRTMSRTM(*m*))，有以下属性成立：

[*Q*](*l*,*e*,*t.*(*ttb*,*te*]) Write(*J*,*l*,*e*)@*t*)

(*t’*,*e’*.((*t’tb*,*te*])Send(*I*,*e’*)@*t’*)(*e*,*tR.*(*tR<t*’)

(Read(*I*,*m.pcr.s*,*e’’*) @*tR*)*e’*=*SIGAIK*(*m*)*-1*){|*PCR*(*s*),*e’’*|})

该属性表明在验证过程中如果本地没有写入内存的操作且发送了数据*e*'，则在之前的某时刻本地一定读取了值*e*"，且*e*'是一个签名值。利用推理规则SEQ和公理Act1证明上述不变量成立。利用诚实规则并进行简化后可得：

[Verifier(*m*)]*tR*,*e*,*I.*(*tR<te*)= (*m*) 

Contain(*e*,*SIGAIK*(*m*)*-1*){|*PCR*(*s*), *seq*(*BIOS*(*m*),*OSLoader*(*m*),

*VMM*(*m*), *Dom0\_Kernel*(*m*), *vTPM Builder*(*m*),

*vTPM-VM Binding*(*m*), *VM Builder*(*m*))|})

(Read(*I*,*m.pcr.s*,*e’’*) @*tR*)*e*=*SIGAIK*(*vm*)*-1*){|*PCR*(*s*),*e’’*|}

分别利用等值公里Eq和Read公理，有

[Verifier(*m*)]*tR*,*e’’*,*I.*(*tR<te*)

Contain(*e*,*SIGAIK*(*m*)*|-1*{|*SIGAIK*(*m*)*|-1*{|*PCR*(*s*),*e’’|*},

*seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*), *Dom0\_Kernel*(*m*),

*vTPM Builder*(*m*), *vTPM-VM Binding*(*m*),

*VM Builder*(*m*)))|})(Mem(*m.pcr.s*,*e’’*)@t*R*) (**14**)

此时需要判定*e’’*的值，根据上述推理过程可知有两种可能：

(*e’’= seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),

*Dom0\_Kernel*(*m*),*vTPM Builder*(*m*), *vTPM-VM Binding*(*m*), *VM Builder*(*m*))) Contain(*e*,*SIGAIK*(*m*)*-1*){|*PCR*(*s*),

*seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),*Dom0\_Kernel*(*m*), *vTPM Builder*(*m*), *vTPM-VM Binding*(*m*),

*VM Builder*(*m*))|})  **(15)**

根据公理PCRC：

├(Mem(*m.pcr.s*,*e’’*)@*t*) Contains(*e*,*SIGK*{|*e’*|})

以及Mem(*m.pcr.s*,*e’’*)存在的事实，可知式(14)中第2种可能不成立，故只有

*e’’= seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),

*Dom0\_Kernel*(*m*), *vTPM Builder*(*m*),

*vTPM-VM Binding*(*m*), *VM Builder*(*m*))成立。

利用等值公理Eq对式(14)进行变换可得

[Verifier(*m*)]*tR*,*.*(*tR<te*)

(Mem(*m.pcr.s*, *seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),

*Dom0\_ Kernel*(*m*), *vTPM Builder*(*m*),

*vTPM-VM Binding*(*m*), *VM Builder*(*m*))@tR)

即定理2属性式(12)得证。利用属性式(12)结论及定义1，可直接证明属性式(13)成立。

根据上述证明可知，在TVP-QT信任链构建过程中，能够有条件地保持其信任属性，即构建信任链所需要执行的不同程序在跳转过程中，不会被其他恶意代码所控制或插入，从而不存在信任缺失的情况，且这种信任属性能够向远程验证者提供证明。

5.1.3 **可信衔接点TJP的本地验证及远程证明**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

5.1.3.1 本地程序执行

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××根据3.2节对TVP-QT中TJP信任属性TPTJP定义以及TPvRT中对TCTJP的定义，其信任链本地执行过程中涉及到的程序如图6所示。

*LatelaunchDTRM*(*vTPM-Builder*)≡*vtb*=read *m.vTPM-Builder\_loc*;

Extend *m.dpcr.d*,*m*;

Jump *vtb*

...

*vTPM-Builder*(*TJP*) ≡ *vvb* = read *m.vTPM-VM-Binding\_loc;*

Extend *m.dpcr.d*,*vvb*;

Jump *vvb*

*vTPM-VM Binding*(*TJP*) ≡ *vvmb* = read *m.VM-Builder\_loc*

Extend *m.dpcr.d*,*vvmb*;

Jump *vvmb*

*VM-Builder*(*TJP*) ≡ *vtpmb*= read *m.vTPM\_loc*

Extend *m.dpcr.d*,*vtpmB*;

Jump *vtpmb*

*vTPM*(*m*) ≡ ...

1. TVP-QT中 TJP 信任链传递

程序执行流程：首先确保TJP的vTPM-Builder能正常加载。然后利用DRTM度量TJP的三个组件vTPM Builder、vTPM-VM Binding、VM Builder，从主机内存地址中读取vTPM Builder的代码，将其扩展到一个PCR中（其中，*m.pcr.d*表示TJP的度量值存储与动态度量的PCR中）；之后执行命令Jump *vtb*将控制权交给vTPM Builder，按照上面的过程依次度量vTPM-VM Binding、VM Builder。

在此过程中，对TJP的动态度量必须在m启动之后且创建vm之前，否则会导致TJP无法按顺序正确度量。将其表示为

Honest(TPMSRTM (*m*)TJPDRTM(*m*)

TJPDRTM(*m*)vTPMSRTM/DRTM (vm))。

此外，TVP在启动m时，相应的线程K对必须能够对当前m对应的PCR值有锁控制，这种控制对潜在的攻击者也成立，表示为

ProtectedSRTM(*m*)=*t*,*K.*(Reset(m,*K*)@*t*)(IsLocked((*m.pcr.s*,*m.pcr.d*),*K*)@)*t*

由于TJP 的vTPM Builder被抽象为一个单独的应用程序（无论其实现形式是独立的轻量级可信执行环境或仅是提供可信功能的应用进程），利用Latelaunch(vTPM Builder)动态加载机制确保其可信执行，即*K*DRTM成立[27]。

5.1.3.2 本地可信属性描述

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××基于定义2及TJP度量后的PCR和其中的每个组件存在的唯一性、确定性映射关系，可将TJP的本地信任传递属性归纳为：如果最终的PCR中度量值序列是正确的值，那么TJP信任链所加载的程序顺序就是正确的。即TJP的本地信任传递属性就是要求所有相应启动程序如vTPM Builder、vTPM-VM Binding、VM Builder等都能按确定的先后顺序加载。以LS2将这种顺序形式化表示为

MeasuredBootDRTM(TJP,*t*)=



(Jump(*K*,*vTPM Builder*(*TJP*)))@*tvtb*)

(Jump(*K*,*vTPM-VM Binding*(*TJP*))@*tvvb*)

(Jump(*K*,*VM Builder*(*TJP*))@*tvvmb*)

(Jump(*K*,*vTPM*(*TJP*))@*tvtpmb*)

(┐Reset(*m*)on(*tvtb*,*t*])(┐Jump(*K*)on(*tvtb*,*tvvb*))

(┐Jump(*K*)on(*tvvb*,*tvvmb*))(┐Jump(*K*)on(*tvvmb*,*tvtpmb*))

上述公式表示：如果TVP基于信任链构建TJP信任环境，则其启动过程一定是从vTPM Builder跳转到vTPM-VM Binding，然后到VM Builder，而在此期间不会有其他程序执行。这就需要证明上述程序启动序列与PCR值之间的一一映射关系。基于前文的假定前提，要证明的信任链本地信任属性如下。

**定理3** 如果TJP加载成功，且与该TJP加载过程对应的PCR值为*seq*(*vTPM Builder*(*TJP*),*vTPM-VM Binding*(*TJP*),*VM Builder*(*TJP*))，那么该TJP的本地信任链传递过程就是唯一的、正确的，即确定地从vTPM Builder(*TJP*)到vTPM-VM Binding(*TJP*)再到VM Builder(*TJP*)。该信任属性形式化表示为

ProtectedDRTM(TJP)+

Mem(*m.dpcr.d*,*seq*(*vTPM Builder*(*TJP*),

*vTPM-VM Binding*(*TJP*)),*VM Builder*(*TJP*))

MeasuredBootDRTM(TJP,*t*)

证明过程类似m的信任链本地可信属性的证明，在此不再叙述。

5.1.3.3 信任链远程验证

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××TVP-QT的TJP 需要向外部挑战者证明自己所声称的信任属性，即其信任链传递过程中所执行程序的确定序列，使外部挑战者相信它的确按上述信任链构建了可信执行环境，需要证明MeasuredBootDRTM(TJP,*t*)成立。

**a.远程验证程序执行**

首先，根据 TCG 远程证明协议规范及在虚拟化平台中的实现，给出TJP 信任传递的远程验证过程中涉及到的程序，如图7 所示。

*TPMDRTM*(*TJP*) ≡ *w =* read *m.pcr.d*;

*r =* sign(*PCR*(*s*),*w*),*AIK-1*(*m*);

send *r*

*Verifier*(*TJP*)≡ *sig =* recieve;

*v =* verify *sig*, *AIK*(*m*);

match *v* ,(*PCR*(*s*),

*seq*(*vTPM Builder*(*TJP*),*vTPM-VM Binding*(*TJP*),*VM Builder*(*TJP*))

1. TVP-QT中m信任传递的远程验证程序
2. The remote authentication of TJP’s trust chain

首先，m读取本地TJP的PCR值，用AIK签名（*AIK-1*(*m*)）并将其发送给挑战者。然后，挑战者验证该签名，并用预期的度量值序列与收到的值进行对比，如果匹配，则表明该主机m的TJP拥有所声称的可信属性，否则验证失败。在此过程中远程验证者与TJP所属的主机m应是不同实体，以保证该验证过程的有效性。

这些前提条件形式化表示为

DRTM = {Honest(*AIK*(m))),*AIK*(*m*)}

**b.信任链属性的远程验证**

根据远程证明协议执行流程，给出以下信任传递属性的远程证明目标。

**定理 4**如果远程验证者确认TJP提供的度量值是唯一的、正确的，那么该TJP对应的PCR值一定是如下的确定序列*seq*(*vTPM Builder*(*TJP*),*vTPM-VM Binding*(*TJP*),*VM Builder*(*TJP*))，因为根据定理3可知，该序列表明该虚拟机的确执行了相应的信任链传递过程。

形式化表示为：

DRTM├ [Verifier(*m*)]*t.*(*t<te*)

(Mem(*m.pcr.d*,*seq*(*vTPM Builder*(*TJP*),

*vTPM-VMBinding*(*TJP*),*VM Builder*(*TJP*))@*t*) (**16**)

DRTM，ProtectedSRTM(*m*)

├[Verifier(*m*)]*t.*(*t<tE*)

MeasureBootDRTM(*m*,*t*) (**17**)

证明过程类似m的信任链远程验证的证明，在此不再叙述。

5.1.3 **虚拟机vm的本地验证及远程证明**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

5.1.3.1 本地程序执行

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

5.1.3.2 本地可信属性描述

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

5.1.3.3 信任链远程验证

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

1. 远程验证程序执行

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

1. 任链属性的远程验证

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

**5.2 基于扩展无干扰理论的TVP-QT分析**

5.2.1 **扩展无干扰理论基本假定及定义**

×××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××遵从Rushby等人的描述，本文以自动机（或叫状态机）的形式，并在安全域、动作等对无干扰进行扩充，在动作主体、动作主体对安全域和系统状态的影响扩展到无干扰理论中，给出无干扰的基本定义如下。

**定义1.** 系统M由如下要素构成：

1. 一个包含唯一初始状态的状态集。约定使用等表示系统状态；
2. 一个由系统中所有原子动作组成的动作集，约定用等表示原子动作；
3. 一个由系统中所有发出原子动作的动作主体集，每一个主体可能发出不同的动作，相同的动作也可能由不同的主体发出。即每一个属于中的动作都有一个包含于中的主体集。即动作的主体集。
4. 一个由系统中所有行为构成的行为集。其中，行为表达为原子动作序列的形式，约定用希腊字母等表示行为。一个行为的示例是，其中是连接符；
5. 一个输出集，其中包含了使用动作进行观察时所看到的结果；
6. 每一个原子动作以及原子动作的主体都有自身所属的并且不可再分的Min安全域，这些安全域构成的集合称为Min安全域集；安全域中的主体向系统发出操作动作与系统进行交互，并且能够观察到相应的结果。安全域的划分可以限制系统中的信息流动。并且根据云计算环境运行机制，安全域集的某一子集也可能单独成为运行的一个组合安全域，这些由子集组成的组合安全域称为，且中任意两个元素可能存在交集。即对于，有。且对应与系统状态，域也有相应的状态，用表示域状态集合，约定使用分别表示安全域的状态在系统状态为本安全域状态，显然有

。

1. 安全策略和。安全域集之间可以有信息流动，信息是否能够在特定域间流动由安全策略和决定，和分别称为干扰和无干扰关系，两者互为补集；
2. 动作主体到动作的映射函数：。返回一个特点动作所属的动作主体；
3. 安全域到动作的映射函数：。返回一个特定动作所属的Min安全域，并且必然，使得。
4. 单步系统状态函数：。.单步函数描述的是系从前一个状态，由某一安全域某一动作主体执行某个动作之后，应该到达的后一个状态；在系统存在安全域中的主体，执行动作之后，对系统的改变为。同理，动作对动作所属安全域的改变单步域状态函数为，在安全域在下的主体，执行动作之后，对安全域改变为。

且单步状态函数满足以下条件：

。

1. 行为结果函数：。行为结果函数给出了：在状态使用特定的动作所观察到的行为执行结果。
2. 行为执行函数：。如果用表示空动作序列，则可以表示为右递归的形式：

对于系统状态：



对于安全域状态：

。

需要强调的是，无干扰模型将输入定义为行为，也就是原子动作的连接序列。本文遵从原有的无干扰理论，对系统进行描述。

**定义2** .系统视图。

系统视图对应于实际机器M，其关注于存储单元(内外存单元、芯片存储单元等)及其取值，以及可观察存储单元、可修改存储单元等内涵。

1. 存储单元集。机器的每一个存储单元都有一个名字。所有存储单元名字的集合构成存储单元集，又叫做名字集。
2. 系统视图值集。每一个存储单元在特定的状态都会有一个特定的值。具体的取值可以由下面的内容函数计算。所有取值的集合构成值集。
3. 系统视图内容函数。

**定义3** .域视图。

域视图对应于M中的安全域，是M视图的子集。

对应于系统状态等相关内容函数，域视图有以下表示：

1. 域视图值集。安全域中每一个存储单元在特定的状态都会有一个特定的值。具体的取值可以由下面的域视图内容函数计算。所有取值的集合构成值集。
2. 域视图内容函数。
3. 域视图观察函数和域视图修改函数。观察函数和修改函数分别给出了特定的安全域所能观察和修改的存储单元的集合，其中是幂集计算。

**定义4** .主体视图。

1. 在整个系统M中，域中有很多可以发出动作的主体，即

。

亦有。

仅次于安全域的单位为动作主体，动作主体对应系统中的存储单元在域不同状态下，由动作发出后，其存储单元也会改变。

主体视图值集。动作主体每一个存储单元在特定的状态下都会有一个特定的值，其中有。具体的取值可以由下面的主体视图内容函数计算。

1. 主体视图内容函数。
2. 主体视图观察函数和主体视图函数。不同的主体能观察和修改的存储单元的集合不同。

**定义5.**关系称为是等价关系，当且仅当同时满足输出一致性和（弱）单步一致性。

1. 输出一致性：。

在系统视图上：。

并且由可得域视图的输出一致性：

。

1. （弱）单步一致性：

对于传递安全策略，需满足单步一致性：

。

对于非传递安全策略，需满足弱单步一致性：

。

与单步一致性相比，弱单步一致性增加了条件，这个条件很重要。这是因为：弱单步一致性对应的是非传递无干扰，因此其除了要考虑直接干扰关系之外，还要考虑间接干扰关系上的“单步一致性”。注意：本文中遵从Rushby的符号表达，约定使用表达蕴含关系。

**定义6.**间接干扰关系。

对于非传递安全策略而言，安全域虽然不能直接干扰安全域（因为），但是，仍然可以间接对进行干扰(因为)。若将定义1当中的干扰关系称作是“直接干扰关系”，则可以定义“间接干扰关系”如下:



**定义7.**干扰源集。

干扰源集的递归定义如下:

，且



，且。其中，。

干扰源集的含义是:抽取所有对安全域有直接或间接干扰关系的安全域形成集合。

**定义8.**弱预期函数。

，且

。

其中，，

。

弱预期函数的含义是，“将所有对安全域有直接或间接的干扰关系的动作保留，并将除此以外的所有动作删除”，从而得到在非传递无干扰安全策略控制下的预期行为。 **定义9.**域集等价关系:。

并且Rushby定义了系统满足非传递性无干扰策略的判定定理。

**定理1 .**系统满足非传递性无干扰策略的判定定理。

设M 是一个视图隔离的系统，有一个具有非传递性的～>策略，并且M满足：输出一致性、弱单步一致性和局部干扰性，则M 满足非传递性无干扰策略。

5.2.2 **TVP-QT信任链传递形式化描述**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××由可信的定义可以看出，可信的核心内容在于组件行为的可预测性。组件行为各有不同：对应用程序来说，它的行为包括输出屏幕信息、显示图片、发出声音、创建网络连接等等；对内核来说，它的行为包括产生进程、管理内存、访问控制等等。这些不同的行为都符合有限状态自动机模型的抽象描述：一个状态确定的系统，其行为是可以预测的，即一个固定的输入队列可以有一个固定的输出队列，也就是这些行为都可以通过信息系统的状态和输出来表现。而且，从组件外部来看，它们只能通过输出来表现。

因此根据可信的定义，一个初始状态确定、输入确定的有限状态自动机系统是一个可信系统。但在现实系统中，组件的状态除了取决于自身（通常是代码、堆栈、数据），还受其所处环境的影响（包括硬件平台、操作系统、系统中其它的组件的影响），也就是说组件不是一个状态孤立的有限状态自动机，而是一个部分状态受其它关联组件控制的状态机。系统是否可信由其组件和组件之间的交互所决定。这里的组件可以是一个硬件模块、软件模块、应用程序等。组件的可信性可以由图-的模型表示，模型包括4个元素：组件、输入、输出和组件间的干扰。组件运行是否可信，用组件输出是否符合预期来表示，因此一个组件运行可信取决于3个因素：

（1）完整性因素。即组件本身没有被篡改过；

（2）输入因素。组件的输入在允许范围内；

（3）干扰因素。其它组件对其状态没有干扰。



图1 组件间干扰造成运行不符合预期示意

可见：如果组件本身没有被篡改，并且组件间不存在干扰，则组件的输出符合预期，系统中组件的运行是可信的。所以，如果要使信任链传递有效，必须消除组件之间的干扰。

本文基于扩展后的无干扰理论，对TVP-QT信任链模型进行形式化描述。由于扩展后的无干扰理论从系统主图、域视图、主体行为视图由一个整体系统向系统内的主体进行细分，本文针对TVP-QT信任链传递模型，从以下方面对信任链进行形式化描述。

**定义12：**信任关系可以用以下二元关系表示：（针对于安全域级），（针对于组合安全域级），（针对于动作主体级）。，若，则表示域对域进行了可信度量，并且度量成功，域对域是信任的，然后进行控制权的转移。

首先，按照Min安全域的信任传递方式，可以表示以下式子：



其中表示为可信虚拟平台的可信度量根：即图2中TPM下的CRTM。

按照组合安全域的信任链传递方式，可以表示为以下公式：



基于可信计算技术的装载前度量技术实现的可信链可以这样形式化地描述：

；

表示如果由安全域通过摘要运算获得的摘要值与预期值相等,则安全域信任组件,信任关系将由传递至系统控制权也转移到其中digest(A,B)表示安全域A对安全域B进行摘要运算的结果，expect(A)表示组件A的完整性预期值，表示控制权从A转移到B。

同理对应组合域有：

。

5.2.3 **扩展无干扰信任传递判定定理**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××单纯的通过完整性验证实现的信任链传递是否有效无法进行验证。只有当系统具有特定的安全机制，满足一定的安全策略，组成系统的各安全域之间的信息流动受到一定安全策略限制，使得组件的运行不受干扰，这时，用完整性度量方法所建立的信任链才是有效的。非传递无干扰关系描述的是一种隔离性要求比较严格的通道控制安全策略，具有非传递无干扰关系的系统组件之间只有直接干扰关系，不存在间接造成的干扰关系。信任链传递模型关键之处是验证系统中是否满足非传递无干扰关系，但从非传递无干扰关系的定义出发很难进行验证，本文给出无干扰信任传递判定定理，用于判定可观测的系统状态和输出在满足什么条件时，信任链的建立和传递才是有效的。

下面本文分别从系统、安全域（和）、动作主体三个角度对本文提出的信任链模型应该满足的安全策略进行描述和分析验证。其中，从系统角度描述系统符合的安全策略，则需要对安全域的输入和系统状态的论述。同理，从安全域角度描述安全策略，则需要对动作主体的输入和对系统状态的进行论述。

**定理2** TVP-QT系统满足非传递无干扰关系的判定定理。

1. 系统的域满足输出一致性。即一个内部操作动作造成的输出影响只依赖于发出动作域的系统视图。且满足一下条件：



1. 相对于（1），系统的动作主体满足输出一致性。即一个安全域内的操作动作造成的输出影响只依赖于发出动作动作主体的系统视图。

；



1. 系统中发生的一个动作造成的对系统状态影响只与发出该动作的域的上一状态系统视图相关联。

；



1. 相对于（3），系统中发生的一个动作造成的对安全域状态影响只与发出该动作的动作主体的上一状态系统视图相关联。

；

1. 系统中，如果一个动作改变了其动作主体的值，则发出该动作的主体一定可以写、访问该主体的系统视图，并且可以写、访问该动作所在的域的系统状态。

；



1. 系统内对系统内的系统状态的操作有以下关系。



1. 系统内若发出某一动作，则该动作的动作主体，所属域唯一确定的。

，在系统状态，必然存在，，可唯一确定。

证明：

根据无干扰给出的定理1，系统需要满足输出一致性，单步一致性，局部干扰性即可。

（1）证明输出一致性：

（2）证明单步一致性：

需要证明：

①

②

式②是在域状态下的单步一致性，是在式①动作主体在域状态的延伸，本文在此只证明式①。

，存在下式：



若，由定理2条件（2）可得：

；

若，由定理2条件（2）可得：

，

其他情况下：；

又由也可得



（2）证明单步一致性：

即证明：

亦可从其逆否命题证明：

，。

由定理2条件（3）可得

，

由定理2，可得成立。

（3）对动作确定性的证明。

上述给出了一个计算机系统满足非传递性无干扰关系的形式化规范，也给出了一种判定可信计算平台信任链传递关系的有效方法。

5.2.4 **基于扩展无干扰的TVP-QT验证**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

我们基于开源的虚拟机监视器( VMM) 系统Xen，结合本文第三节介绍的TVP-QT平台，利用虚拟隔离实现了一个满足非传递无干扰的系统。它将应用完全隔离，各应用之间不能直接共享信息，所有隔离域之间的信息交换均通过虚拟机监视器进行。并在此平台下，验证了TVP-QT信任链是符合非传递无干扰判定定理。

（http://wangzan18.blog.51cto.com/8021085/1727106备用）

如图所示，从TVP-QT第一层到系统最上层存在着很多不同的组合安全域，比如TPM域、VMM域等，并且针对系统中的可信衔接点CJP的不同组件是存在于不同的组合安全域中。由CJP存在域不同的安全域中可知本文对安全域和组合安全域的划分是正确的。并且CJP中的某一组件若发出动作，可能存在与Dom0域也有可能存在vTPM Manager域中。并且Dom0中的组件主体发出的某一动作可能会与信任链信息流传递时与CJP中发出的某一动作相同，但是在对信任链是无干扰的，验证了在系统上对动作发出主体的定义的有效性。

由于TVP-QT系统之上可以并行运行虚拟机，根据信息流的不同情况，本文对实际系统的无干扰验证分为两种情况进行验证。

**情况1：存在一个用户虚拟机**

在此种情况下虚拟机运行在VMM之上，VMM系统上的设备驱动模型由运行于虚拟机上的前端驱动和运行于VMM之上的后端驱动组成，实现对硬件功能的使用。该用户虚拟机在运行过程中产生的信息流可以从图中的顺序进行传递。若使该虚拟机信任链有效，必须确保系统中存在非预期的干扰。

依据定理2，该系统中的I/O设备驱动程序满足如下要求：

1. 由输出一致性的定义可知，该虚拟机使用的虚拟资源可以由VMM进行分配，并且不存在其他虚拟机无干扰的动作存在。
2. 由局部干扰性定义可知，VMM系统中，虚拟I/O设备除其所在的虚拟机外其它虚拟机不能改变它的运行状态，I/O设备驱动与虚拟设备间所传输的数据对其它虚拟机是不可见和不可修改的。该VMM系统的隔离机制确保虚拟机必须采用虚拟设备接口访问后端驱动程序。
3. 由弱单步一致性可知，该虚拟机在运行过程中的动作只会对此系统进行状态的改变。

**情况2：存在多个用户虚拟机**

该情况下，在存在一个用户虚拟机之上在运行多个虚拟机，多个虚拟机需要共享设备资源，仅仅对在情况1下增加了虚拟机之间的信息流的无干扰情况。因此本文对多个虚拟机在系统中共享资源的同时进行无干扰的验证。

如图为多个用户虚拟机IO操作示意图：



图 多个用户虚拟机IO操作示意图

（1）由输出一致性的定义可知，VMM维护的虚拟资源必须具有其属于哪个虚拟机的属性标识。可以保证系统中的输出信息可以定位到某一个虚拟机，此虚拟机的不同状态下产生的操作不干扰其他虚拟机的运行。

（2）由局部干扰性定义可知，VMM系统中每个虚拟机能够访问的I/O寄存器被限制，能够禁止未授权的访问。

（3）由弱单步一致性定义可知，对于若干虚拟机共享的客体对象，VMM系统必须具有同步保护机制以防止不同虚拟机对该资源的竞争。

该系统依据非传递无干扰策略模型对传统VMM的虚拟I/O设备体系进行了改造，只要根据应用程序需求，合理划分安全域，各应用安全域的运行不受其它应用程序干扰，依据本文给出的TVP-QT信任传递模型，经过完整性验证，系统运行能够达到可信目标。

**5.3 本章小结**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××文首先建立一个具有瀑布特征的可信云环境信任链模型，然后按照云计算环境运行特征，对原有无干扰理论中的安全域、动作等定义进行扩充，并将动作主体和动作对安全域以及系统状态的影响等扩展到无干扰理论中；最后应用此扩展的无干扰理论来分析可信云环境信任链传递模型，用形式化的方法证明当符合非传递无干扰安全策略时，云环境安全域之间的信息流受到安全策略限制，隔离了域之间的干扰，满足此条件时用完整性度量方法所建立的云环境信任链才是可信的、有效的。最后本文基于Xen系统实现了信任链模型，并利用无干扰理论对该信任链模型进行了分析和验证，证明了扩展后的无干扰理论验证信任链模型的有效性。

**6** 总结与展望

**6.1 工作总结**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

**6.2 研究展望**

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

参考文献

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

致谢

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

在校期间的科研成果

××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××××

1. [] yyy [↑](#endnote-ref-0)
2. [] ；；； [↑](#endnote-ref-1)
3. [] ooo [↑](#endnote-ref-2)