一种具有瀑布特征的可信虚拟平台信任链模型

XX[1]，XX[1,2]

(1.XXXXXX XXXXXXX，XX XX，XXXXXX;
2.XXXXXX XXXXXXXX，XX)

摘　要　将虚拟化技术与可信计算相结合构建的可信虚拟平台及其信任链模型是目前的一个研究热点。目前大部分的研究成果是采用在虚拟平台上扩展传统信任链的构建方法，不仅模型过粗且逻辑不完全合理，而且还存在底层虚拟化平台和顶层用户虚拟机两条分离的信任链问题。为此，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。不仅从理论上证明了该模型的正确性，而且对实例系统的分析和讨论也表明了该模型的通用性与可行性，并在 Xen 中对该模型进行了仿真实验，实验结果表明本信任链传递理论可以保证可信虚拟化环境在整个运行过程是安全可信的。

关键词　虚拟化；可信计算；虚拟可信平台；信任链；可信衔接点

Research on the Trust Chain Model with Waterfall Characteristic Based on the Trusted Virtualization Platform

XXXX[1], XXXX[1,2]

(1.xxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxx,xxxxxxxxxxxxxxxxxxxxxx, xxxxxxx;
2.xxxxxxx　xxxxxxx xxxxxxx,xxxxxxx xxxxxxx xxxxxxx, xxxxxxx, xxxxxxx,xxxxxxx)

Abstract The trusted virtual platform which is constructed by the combination of virtualization technology and trusted computing and its trust chain have become one of the key focuses in the researched fields. But at present, most of the researching achievements construct the trust chain by extending the conventional trust chain model, as a result, the model is not precise and the logic is not completely reasonable. Moreover, two separate trust chains have existed, one is starting of the underlying virtual platform, and the other is the starting of the top-level user virtual machine. In order to solve this problem, this paper proposes a model of the trust chain with waterfall characteristic called TVP-QT for the trusted virtual platform. This model starts with the physical TPM, and increases a Trusted-Joint Point called TJP between the chain of the underlying virtual platform and the chain of the top-level user VM.The TJP is in charge of the measurement of vTPM for VM after the trusted chain is transmitted from the underlying virtual platform to the TJP, and then the vTPM gets the control, and is in charge of the measurement of the related components and applications of the top-level user VM in the starting process.The TJP which has the waterfall characteristic between the underlying virtual platform and the top-level user VM can be viewed as a connecting link, and it can satisfy with the hierarchical and dynamic characteristics of the virtual platform, moreover guarantee the trust of the whole virtual platform. Finally, we not only have

———————————————

本课题得到xxxxxxxx（xxxxxxxx，xxxxxxxx，xxxxxxxx）；xxxxxxxx(xxxxxxxx)；xxxxxxxx（xxxxxxxx）资助。

作者简介: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

proved correctness of the model in theory, but also have analyzed and discussed the generality and feasibility of the proposed trust chain model in the instantiation system. And then the simulation experiment based on Xen is realized, and the experiment results show that the trust chain can ensure the trust and credibility of the trusted cloud platform in the whole running process.

Key words: Virtualization; Trusted Computing; Trusted virtual Platform; Trust Chain; Trusted-Joint Point;

1 前言

虚拟化技术因具有节省成本、提高效率等特有优势使其得以快速应用推广，比如，在云计算等大型计算应用环境中，虚拟化平台已经成为承担海量计算和应用服务的基础。但随之而来的一个关键问题就是如何为虚拟化平台提供服务可信的保障[1-3]，用户在使用虚拟化平台提供的资源和服务时，亟需确认该服务平台是否可信任[4-6]。可信计算技术基于硬件信任根，能够为平台构建从底层硬件到上层应用程序的信任链[7][8]，并结合度量与远程证明机制为外部提供可信证明[9-11]，从而为平台提供可信运行环境保障，因此，利用可信计算技术构建可信虚拟平台（Trusted Virtualization Platform, TVP）环境并对其构建信任链模型成为目前的研究热点。

自从 Stefan Berger[12]等人提出 TVP 概念以来，文献[13-16]等在如何构建针对具体应用场景的 TVP 功能应用以及构建统一、抽象的 TVP 概念方面都做了大量工作，取得了较好的成果，并达成了一些基本共识。目前，研究此方面的学者绝大多数都认为，TVP 在物理上体现为一个支持虚拟化技术的可信主机，它与普通可信计算平台的区别主要在两方面，其一是拥有构建于硬件可信芯片可信平台模块（Trusted Platform Module, TPM）基础上的虚拟信任根；其二是并发地为多个客户应用虚拟机（Virtual Machine, VM）提供信任环境。其基本的运行架构如图 1 所示。从功能上看，TVP 主要分为 4 个层次，硬件信任根 TPM 作为最底层，是平台信任的物理保障。第二层主要包括虚拟机监视器（Virtual Machine Monitor, VMM）及管理域（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。

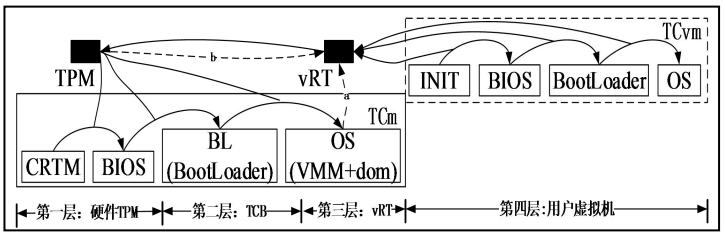

图 1 TVP 基本运行架构

尽管如此，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。

针对上述问题，本文提出了一种具有瀑布特征的 TVP 架构 TVP-QT，并对 TVP-QT 信任链传递模型进行了构建，建立了拥有可信衔接点（Trusted-Joint Point, TJP）的 TVP 及其完整的信任链模型。该模型以虚拟化硬件层物理 TPM 为起点，在可信虚拟化平台信任链和可信虚拟机信任链之

间加入可信衔接点。当信任链从可信云平台传递到可信衔接点时，由可信衔接点负责对可信虚拟机的vTPM及相关组件进行度量，之后再将控制权交给虚拟可信平台模块（Virtualization Trusted Platform Module, vTPM），由vTPM负责对虚拟机的VBIOS，VMOS到VM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，能满足云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

2 相关工作

目前针对TVP及其抽象模型以及信任链传递模型的研究得到了国内外学者的广泛深入和研究，本文就目前对TVP、TVP信任链模型的研究进行了以下总结和分析。

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，它们的主要思想都是将计算平台分为可信区域与不可信区域，可信区域上运行高安全需求的VM。随后，Berger[12]等人首先提出构建TVP的基本组件vRT、vTPM等基本思想，并且构建了具体的TVP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]、王丽娜[16]等人把TVP在云计算环境中加以应用，以保护虚拟机运行环境及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。

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。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MD_n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MD_n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。

综上所述，对于可信虚拟化平台 TVP，无论 TVP 基本运行架构以及信任链传递模型均存在问题。特别地，由于 TVP 具有层次性和动态性，已有的研究成果不能精细的描述虚拟化环境中复杂的信任链传递机制，且存在两条分离的信任链问题。

3 具有瀑布特征的 TVP 及信任链模型

为解决上文提出的问题，本文提出了具有瀑布特征的 TVP-QT 及信任链模型，本节将对 TVP-QT 及信任链模型进行详细的描述。本文主要针对 TVP-QT 及信任链模型，因此不考虑虚拟化平台自身的固有安全机制，比如虚拟机监控器的特权操作、虚拟机之间的隔离及内存操作控制等，可参考 GillesBarthe 等给出的形式化描述与分析[32]。本文在本节对 TVP-QT 的功能组件以及 TVP-QT 信任链信任属性进行定义，本文在下一节将利用文献[27]提出的安全逻辑形式化方法对信任链进行形式化分析。

3.1 TVP-QT信任模型

我们基于已有的 TVP 研究方案，提出了 TVP-QT 信任模型，如图 2 所示。

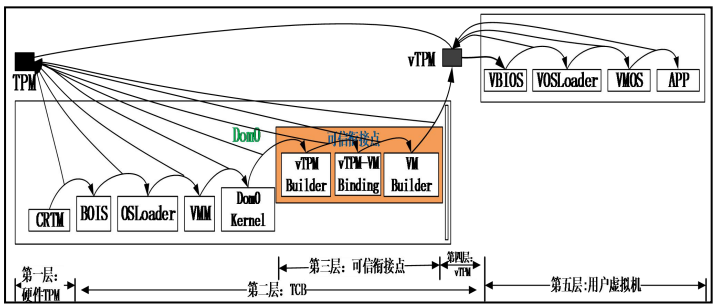

图 2 TVP-QT 运行架构

从功能上看 TVP-QT 运行架构可以主要分为 5 个层次，硬件信任根 TPM 可作为虚拟化平台的底层，从物理上保障虚拟化平台；第二层主要包括 VMM 及管理域相关组件，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。最上层为用户虚拟机层次，是可信虚拟化平台上与用户密切相关的虚拟机部分，其运行时组件包括 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。

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:={vm_1, … , vm_n}，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 是硬件信任根，主要用于为物理平台提供信任保障，它拥有非易失存储及密钥存储等固有特性；vRT 包含 TJP 和 vTPM，在功能实现上可表现为 m 中内核组件或独立的可信组件，这里将其抽象为一个独

立功能组件，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), (vRT_1, vm1), … , (vRT_n, vmm)}

= {(TPM, (vmm, Dom0 Kernel, TJP)), ((TJP, $vTPM_1$), vm_1), …,((TJP, $vTPM_n$), vm_n)}

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。

显然，相对于已有的 TVP，我们提出的 TVP-QT 信任模型具有如下特点：

（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信任链及属性

TCG 组织从实体行为预期性角度给出可信的定义，并采用装载前度量的方案，给出了信任链传递和控制权转移的过程。与普通可信计算平台类似，TVP 的信任链同样需要保障平台能够基于信任根，通过逐级的信任传递，构建虚拟化平台的可信运行环境。由于虚拟化平台自身的特殊性，它要求并发地执行多个用户虚拟机实例，使得信任传递会出现多个不同分支，这与可信计算最初构建信任环境的思想并不一致。尽管如此，只要虚拟化平台能够确保信任链构建过程的唯一性、正确性，以及能对任意的外部实体 R 证明确实构建了对应的信任链，那么整个虚拟化平台是可信的。如图 3 所示，从外部实体来看，虚拟化平台仍然满足 TCG 最初建立信任环境的思想。

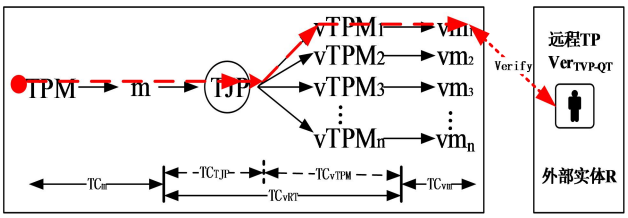

图 3 虚拟化平台可信环境信任链构建与验证

为了确保这种信任传递的正确性，需要对 TVP-QT 信任链进行验证，证明在程序控制权传递过程中，各个进程的确能够按照预期执行，而且能够对外证明上述属性。我们将上述验证目标抽象为信任链的信任属性（Trusted Property, TP），其抽象定义如下。

定义 2 （TVP-QT 信任链模型的信任属性 $TP_{TVP\text{-}QT}$）根据上文对 TVP-QT 信任属性的描述，TVP-QT 的信任属性应该定义为一个二元组 $TP_{TVP\text{-}QT}$:={$TC_{TVP\text{-}QT}$,$Ver_{TVP\text{-}QT}$}，其中 $TC_{TVP\text{-}QT}$ 表示 TVP-QT 信任链模型构建时所包含的可信组件传递序列，即上文对 TVP-QT 信任链模型具体构建过程的描述的各个组件序列。$Ver_{TVP\text{-}QT}$ 表示为对 TVP-QT 信任链模型执行序列的远程认证。

按照 3.1 节对 TVP-QT 中相应功能组件的定义，该 TVP-QT 信任属性可以进一步细分为：

$TP_{TVP\text{-}QT}$:={$TC_{TVP\text{-}QT}$,$Ver_{TVP\text{-}QT}$}

={(TC_m,TC_{vRT},TC_{vm}), (Ver_m, Ver_{vRT}, Ver_{vm}) }

={(TC_m,(TC_{TJP},TC_{vTPM}),TC_{vm}),(Ver_m,(Ver_{TJP},Ver_{vTPM}),Ver_{vm})}

由定义可知 TVP-QT 信任属性可以分为三类：主机 m 的信任属性 TC_m，虚拟信任根 vRT 的信任属性 TC_{vRT}，以及用户虚拟机的信任属性 TC_{vm}。其中 TC_{vRT} 包含 TC_{TJP} 和 TC_{vTPM} 两个属性。下面对 TVP-QT 三类组件的信任属性进行分别阐述。

（1）主机 m 的信任属性表示：

TP_m:={TC_m,Ver_m}

其中，TC_m 表示基于硬件信任根构建的信任链，即主机 m 在本地正确地完成从第一层硬件 TPM 的 CRTM 到 Dom0 Kernel 的可信启动过程：

(CRTM→BIOS→OSLoader→VMM→Dom0 Kernel)$_{TPM\_Static}$

此部分信任链可基于硬件可信芯片 TPM 的可

信度量，且在 TVP-QT 信任链传递过程中不存在除 TVP-QT 信任链组件之外的程序代码加载。Ver_m:=Verify(m, TC_m)表示对外验证主机 m 所声称的信任属性 TC_m，使远程验证者 R 相信 TVP-QT 平台主机 m 拥有这样的信任链属性 TC_m。

（2）vRT 的信任属性为 TP_{vRT}:= {TC_{vRT}, Ver_{vRT}}，表示 vRT 的本地可信加载及其对外的证明。由定义 1 对 vRT 以及定义 2 对 TVP-QT 信任属性的定义，可对 TP_{vRT} 进行进一步细分：

TP_{vRT}:={(TC_{TJP}, Ver_{TJP}), (TC_{vTPM}, Ver_{vTPM})}

其中，TC_{TJP} 表示基于硬件信任根构建的信任链衔接点，其信任传递的过程包括两种情况，其一是在 m 启动时，需要采用静态度量方式对 TJP 进行度量，其信任传递的过程为 ：

(……→Dom0 Kernel→TJP)$_{TPM\_Static}$

完整地表示为：

(CRTM→BIOS→OSLoader→VMM→Dom0 Kernel →vTPM Builder → vTPM-VM Binding→VM Builder ）$_{TPM\_Static}$

其二是在创建 vm 时，为了保证 TJP 的可信（由于 TJP 是应用程序，恶意程序容易篡改），从而使得信任关系可以传递到新建的 vm, 需要采用动态度量方式对 TJP 重新度量验证，信任传递的过程为 (TJP)$_{TPM\_Dynamic}$，完整表示为：

(vTPM Builder→vTPM-VM Binding→VM Builder)$_{TPM\_Dynamic}$

在这两种情况下 Ver_{TJP}:=Verify (TJP, TC_{TJP}) 表示对外验证可信衔接点所声称的信任属性 TC_{TJP}，使远程验证者 R 相信 TVP-QT 的可信衔接点拥有这样的信任链属性 TC_{TJP}；TC_{vTPM} 表示基于硬件信任根构建的用户虚拟机信任根 vTPM，值得注意的是，vTPM 的信任属性与其实现方式密切相关，它可能实现为一个微内核系统或一个应用进程，而且需要建立 vTPM 与 TPM 之间的强依赖关系，以硬件信任根保障 vTPM 的可信。TJP 到 vTPM 的信任传递，既可以采用静态度量，也可以采用动态度量，其信任传递的过程为：(TJP→vTPM)$_{TPM\_Staic}$ 或 (TJP→vTPM)$_{TPM\_Dynamic}$ 。Ver_{vTPM}:=Verify(vTPM, Ver_{vTPM}) 表示对外验证 vTPM 所声称的信任属性 TC_{vTPM}，使远程验证者 R 相信 TVP-QT 的 vTPM 拥有这样的信任链属性 TC_{vTPM}；

（3）用户虚拟机 vm 的信任属性类似主机 m 的信任属性类似，表示为 TP_{vm}:={TC_{vm}, Ver_{vm}}，其中 TC_{vm} 表示基于 vTPM 构建的信任链，在创建 vm 时需采用动态度量方式对 TJP 进行度量，vm 从初始化到应用的可信启动过程：

(TJP)$_{TPM\_Dynamic}$→{INIT→VBIOS→VOSLoader →VMOS→APP}$_{vTPM\_Static}$

Ver_{VM}:=Verify(vm,TC_{vm})表示 vm 信任链的外部验证。

显然，相对于已有的 TVP 信任链模型，我们提出的 TVP-QT 信任链模型具有如下特点：

（1）TVP-QT 信任链模型具有瀑布特征。TJP 将分离的两条信任链链接起来，保证 TVP-QT 信任链构建的连贯性，起到承上启下的作用。

（2） TVP-QT 信任链模型具有动态性和层次性。动态性主要体现在两个方面，其一，从时间上看 ms 的信任链和 vm 的信任链是两条分离的信任链；其二，可信衔接点 TJP 在 ms 启动时采用的是静态度量，而在 vm 创建时，需要动态度量。这是因为为了防止 ms 内的恶意程序对 TJP 进行篡改，破坏新创建 vm 的可信性。层次性主要体现在 ms 的信任链是基础，处于底层，而各 vm 的信任链是信任扩展，处于顶层。底层信任链和顶层信任链通过衔接点 TJP 链接，保证顶层信任链到顶层信任链的信任扩展。

（3）TVP-QT 信任链模型解决了虚拟平台信任链与虚拟机信任链的衔接问题。虚拟化平台存在两条信任链，其一是虚拟平台在启动时的信任链，其二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。这两条信任链如何衔接？已有的 TVP 信任链模型对这个问题没有具体回答，比较笼统；但是 TVP-QT 信任链模型回答得比较具体和清楚。

4 基于扩展 LS² 的 TVP-QT 信任链分析

针对信任链的形式化建模和分析可以确保可信虚拟平台的可验证性。本文将采用已有的形式化分析方法“扩展安全逻辑（Logic of Secure System,LS²）”对 TVP-QT 信任链模型进行形式化分析。对 LS² 的了解请读者参考文献[19][33]。

4.1 基本假定

在对 TVP-QT 信任链属性分析之前本文假定以下条件是满足的：1）TVP 中涉及到的所有系统镜像文件（包括主机 m 及各个用户虚拟机 vm）的完

整性未受破坏，且用户虚拟机已预先植入所需的可信度量及证明代理；2）主机 m 支持动态加载 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。

从 3.2 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。

在这 3 部分中，对（3）的验证分析与文献[19]相同，具体过程读者可参见文献[19]，本文不再论述；下面本文只对（1）（2）进行验证分析。

4.2 m信任链的本地验证及远程证明

4.2.1 本地程序执行

根据 3.2 节对 TVP 中 m 信任属性 TP_m 定义以及 TP_{vRT} 中对 TC_{TJP} 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*) ≡ ...

图 4 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。

4.2.2 本地可信属性描述

根据上述信任链传递中程序执行过程可知，最终体现 m 信任链的是主机度量之后的 PCR 值，它与执行程序之间存在唯一性、确定性映射。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：如果最终的 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 LS^2 将这种顺序形式化表示为

$$\text{MeasuredBoot}_{SRTM}(m,t)=$$
$$\exists t_S.\exists t_b.\exists t_o.\exists t_v.\exists t_d.\exists t_{vb}.\exists t_{vv}.\exists t_{vmb}.\exists t_{o\_app} \wedge$$
$$\exists J.(t_S < t_b < t_o < t_v < t_d < t_{vb} < t_{vv} < t_{vvb} < t_{o\_app} < t)$$
$$\wedge (\text{Reset}(m,J)@t_S) \wedge (\text{Jump}(J,BIOS(m))@t_b)$$
$$\wedge (\text{Jump}(J,OSLoader(m))@t_o) \wedge (\text{Jump}(J,VMM(m))@t_v)$$
$$\wedge (\text{Jump}(J,Dom0\_Kernel(m))@t_d)$$
$$\wedge (\text{Jump}(J,vTPM\text{-}Builder(m))@t_{vb})$$
$$\wedge (\text{Jump}(J,vTPM\text{-}VM\ Binding(m))@t_{vv})$$
$$\wedge (\text{Jump}(J,VM\text{-}Builder(m))@t_{vvb})$$
$$\wedge (\text{Jump}(J,VM\text{-}Builder(m))@t_{o\_app})$$

$\wedge$ (¬ Reset(m)on(t_S,t]) $\wedge$ (¬ Jump(J)on(t_S,t_b))
$\wedge$(¬ Jump(J)on(t_B,t_o)) $\wedge$(¬ Jump(J)on(t_o,t_v))
$\wedge$(¬ Jump(J)on(t_v,t_d))$\wedge$(¬ Jump(J)on(t_d, t_{vb}))
$\wedge$(¬ Jump(J)on(t_{vb}, t_{vv}))(¬ Jump(J)on(t_{vv}, t_{vvb}))
$\wedge$(¬ Jump(J)on(t_{vvb}, t_{o_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*)) $\supset$ MeasuredBoot$_{SRTM}$(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：

$\exists t_S,t_1,t_2,t_3,t_4,t_5,t_6,J.(t_S \le t_1<t_2<t_3<t_4<t_5<t_6<t) \wedge$
(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*) $\wedge$
(Mem(*m.pcr.s*,*seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),
Dom0_Kernel(*m*), *vTPM Builder*(*m*), *vTPM-VM Binding*(*m*) @t_6)
$\wedge$ (Mem(*m.pcr.s*,*seq*(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*),*Dom0_Kernel*(*m*), *vTPM Builder*(*m*), @t_5)
$\wedge$(Mem(*m.pcr.s*, *seq*(*BIOS*(*m*),*OSLoader*(*m*),
VMM(*m*),*Dom0_Kernel*(*m*))@t_4) $\wedge$ (Mem(*m.pcr.s*,
seq(*BIOS*(*m*),*OSLoader*(*m*),*VMM*(*m*)))@t_3)
$\wedge$(Mem(*m.pcr.s*, *seq*(*BIOS*(*m*),*OSLoader*(*m*))
@t_2) $\wedge$ (Mem(*m.pcr.s*, *seq*(*BIOS*(*m*))@t_1) $\wedge$ Reset(m,*J*)
@t_S $\wedge$ (¬ Reset(*m*)on(t_S,*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 LS^2 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。

$$\forall t', b, o$$

$$(((\text{Mem}(m.pcr.s,seq(BIOS,b,o))@t') \wedge (t_s < t' < t)) \supset \exists t_b \cdot ((t_s < t_b < t) \wedge (\text{Jump}(J,b)@t_b))) \wedge$$

$$(\text{IsLocked}(m.pcr.s,J)@t_b) \quad \textbf{(2)}$$

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 LS^2 规则，在某个时间 t_o、t_v、t_d、t_{vb}、t_{vv}、t_{vmb}、t_{o_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)。

$\exists t_S,t_b,t_o,t_v,t_d,t_{vb},t_{vv},t_{vmb},t_{o\_app} \wedge$
$(t_S<t_b<t_o<t_v<t_d<t_{vb}<t_{vv}<t_{vvb}<t_{o\_app}<t)$
$\wedge$ (¬ Reset(*m*,*I*)on (t_S,*t*]) $\wedge$ Reset(*m*,*J*)@ t_S)
$\wedge$ (Jump(*J*,*BIOS*(*m*)) @t_b) (¬ Jump(*J*,*BIOS*(*m*))
on(t_S, t_b))
$\wedge$ (Jump(*J*,*OSLoader*(*m*))@t_o)
$\wedge$ (¬ Jump(*J*,*OSLoader*(*m*)) on(t_b,t_o))
$\wedge$ (Jump(*J*,*VMM*(*m*))@t_v)
$\wedge$ (¬ Jump(*J*,*VMM*(*m*)) on(t_o, t_v))

$\wedge$ (Jump(J,*Dom0_Kernel*(m))@t_d)
$\wedge$ ($\neg$ Jump(J,*Dom0_Kernel*(m)) on(t_v, t_d))
$\wedge$ (Jump(J, *vTPM-Builder*(m) @t_{vb})
$\wedge$ ($\neg$ Jump(J, *vTPM-Builder*(m) on(t_d, t_{vb}))
$\wedge$ (Jump(J, *vTPM-VM Binding*(m))@ t_{vv})
$\wedge$ ($\neg$ Jump(J, *vTPM-Builder*(m) on(t_{vb}, t_{vv}))
$\wedge$ (Jump(J, *VM Binding*(m))@ t_{vmb})
$\wedge$ ($\neg$ Jump(J, *VM Builder*(m) on(t_{vv}, t_{vmb}))
$\wedge$ (Jump(J, *VM Binding*(m))@ t_{o_app})
$\wedge$ ($\neg$ Jump(J, *VM Builder*(m) on(t_{vmb}, t_{o_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。

4.2.3 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 MeasuredBoot$_{SRTM}$(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。

TPM$_{SRTM}$(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))

图 5 TVP-QT 中 m 信任传递的远程验证程序

首先，m 读取本地 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，如果匹配，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

$$\Gamma_{SRTM} = \{ \hat{V} \neq AIK(m), \text{Honest}(AIK(m), \{ \text{TPM}_{SRTM}(m), \text{TPM}_{DRTM}(m) \}) \} \quad \textbf{(11)}$$

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：

$$\Gamma_{SRTM} \vdash [\text{Verifier}(m)]_V^{t_b,t_e} \exists t.(t<t_e) \wedge (\text{Mem}(m.pcr.s, seq(BIOS(m),OSLoader(m), VMM(m),Dom0\_Kernel(m),vTPM\ Builder(m), vTPM\text{-}VM\ Binding(m), VM\ Builder(m))@t) \quad \textbf{(12)}$$

$$\Gamma_{SRTM}, \text{Protected}_{SRTM}(m) \vdash [\text{Verifier}(m)]_V^{t_b,t_e} \exists t.(t<t_e) \wedge \text{MeasureBoot}_{SRTM}(m,t) \quad \textbf{(13)}$$

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$[\text{Verifier}(m)]_V^{t_b,t_e}$，利用公理 *VER* 可得到：

$$[\text{Verifier}(m)]_V^{t_b,t_e} \exists t_f,e,I.(t_f<t_e) \wedge \hat{I} = \hat{AIK}(m) \wedge \text{Contain}(e,SIG_{AIK(m)}^{-1}) \{|PCR(s), seq(BIOS(m),OSLoader(m),VMM(m), Dom0\_Kernel(m),vTPM\ Builder(m), vTPM\text{-}VM\ Binding(m), VM\ Builder(m)))|\}) \wedge (\text{Sent}(I,e)@t_f) \vee \exists l.(\text{Write}(I,l,e)@t_f))$$

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 $Q \in IS(\text{CRTM}_{SRTM}(m))$，有以下属性成立：

$[Q]_I^{t_b,t_e}(\forall l,e,t.(t\in t_b,t_e])\supset \neg \mathrm{Write}(J,l,e)@t)$

$\wedge(\forall t',e'.((t'\in t_b,t_e])\wedge \mathrm{Send}(I,e')@t')\supset(\exists e,t_R.(t_R<t')\wedge$

$(\mathrm{Read}(I,m.pcr.s,e'')@t_R)\wedge e'=SIG_{AIK(m)}^{-1})\{|PCR(s),e''|\})$

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：

$[\mathrm{Verifier}(m)]_V^{t_b,t_e}\exists t_R,e,I.(t_R<t_e)\wedge \hat{I}=\hat{AIK}(m)$

$\wedge \mathrm{Contain}(e,SIG_{AIK(m)}^{-1})\{|PCR(s),$

$seq(BIOS(m),OSLoader(m),$

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

$vTPM\text{-}VM\ Binding(m),VM\ Builder(m))|\})\wedge$

$(\mathrm{Read}(I,m.pcr.s,e'')$

$@t_R)\wedge e=SIG_{AIK(vm)}^{-1})\{|PCR(s),e''|\}$

分别利用等值公里 Eq 和 Read 公理，有

$[\mathrm{Verifier}(m)]_V^{t_b,t_e}\exists t_R,e'',I.(t_R<t_e)\wedge$

$\mathrm{Contain}(e,SIG_{AIK(m)|}^{-1}\{|SIG_{AIK(m)|}^{-1}\{|PCR(s),e''|\},$

$seq(BIOS(m),OSLoader(m),VMM(m),$

$Dom0\_Kernel(m),$

$vTPM\ Builder(m),vTPM\text{-}VM\ Binding(m),$

$VM\ Builder(m)))|\})\wedge(\mathrm{Mem}(m.pcr.s,e'')@t_R)$ **(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：

$(e''=seq(BIOS(m),OSLoader(m),VMM(m),$

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

$vTPM\text{-}VM\ Binding(m),\ VM\ Builder(m)))\vee$

$\mathrm{Contain}(e,SIG_{AIK(m)\text{-}1})\{|PCR(s),$

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

$vTPM\ Builder(m),vTPM\text{-}VM\ Binding(m),$

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

根据公理 PCRC：

$\vdash(\mathrm{Mem}(m.pcr.s,e'')@t)$

$\supset \neg\mathrm{Contains}(e,SIG_K\{|e'|\})$

以及 $\mathrm{Mem}(m.pcr.s,e'')$存在的事实，可知式(14)中第 2 种可能不成立，故只有

$e''=seq(BIOS(m),OSLoader(m),VMM(m),$

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

$vTPM\text{-}VM\ Binding(m),VM\ Builder(m))$成立。

利用等值公理 Eq 对式(14)进行变换可得

$[\mathrm{Verifier}(m)]_V^{t_b,t_e}\exists t_R,.(t_R<t_e)\wedge$

$(\mathrm{Mem}(m.pcr.s,seq(BIOS(m),OSLoader(m),$

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

$vTPM\text{-}VM\ Binding(m),VM\ Builder(m))@t_R)$

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。

4.3 可信衔接点 TJP 的本地验证及远程证明

本节根据 3.2 对 TVP-QT 中的相关定义和说明，对可信衔接点 TJP 的动态度量机制进行本地验证和远程证明的形式化描述。

4.3.1 本地程序执行

根据 3.2 节对 TVP-QT 中 TJP 信任属性 TP_{TJP} 定义以及 TP_{vRT} 中对 TC_{TJP} 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。

```
Latelaunch_DTRM(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)                    ≡ ...
```

图 6 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

$\text{Honest}(\text{TPM}_{\text{SRTM}}(m) \succ \text{TJP}_{\text{DRTM}}(m) \wedge \text{TJP}_{\text{DRTM}}(m) \succ \text{vTPM}_{\text{SRTM/DRTM}}(\text{vm}))$。

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

$\text{ProtectedSRTM}(m) = \forall t,K.(\text{Reset}(\text{m},K)@t) \supset (\text{Is Locked}((m.pcr.s,m.pcr.d),K)@)t$

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]。

4.3.2 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

$$\begin{aligned}&\text{MeasuredBoot}_{\text{DRTM}}(\text{TJP},t)=\\&\exists K.(t_{vtb} < t_{vvb} < t_{vvmb} < t_{vtpmb} < t)\\&\forall I \in m.\exists t_{\text{vtb}}.\exists t_{vvb}.\exists t_{vvmb}.\exists t_{vtpmb}.\\&\wedge(\text{Jump}(K,vTPM\ Builder(TJP)))@t_{vtb})\\&\wedge(\text{Jump}(K,vTPM\text{-}VM\ Binding(TJP))@t_{vvb})\\&\wedge(\text{Jump}(K,VM\ Builder(TJP))@t_{vvmb})\\&\wedge(\text{Jump}(K,vTPM(TJP))@t_{vtpmb})\\&\wedge(\neg\ \text{Reset}(m)\text{on}(t_{vtb},t])(\neg\ \text{Jump}(K)\text{on}(t_{vtb},t_{vvb}))\\&\wedge(\neg\ \text{Jump}(K)\text{on}(t_{vvb},t_{vvmb}))\\&\wedge(\neg\ \text{Jump}(K)\text{on}(t_{vvmb},t_{vtpmb}))\end{aligned}$$

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

$$\begin{aligned}&\text{ProtectedDRTM}(\text{TJP})+\\&\text{Mem}(m.dpcr.d,seq(vTPM\ Builder(TJP),\\&vTPM\text{-}VM\ Binding(TJP)),VM\ Builder(TJP))\\&\supset \text{MeasuredBoot}_{\text{DRTM}}(\text{TJP},t)\end{aligned}$$

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。

4.3.3 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 $\text{MeasuredBoot}_{\text{DRTM}}(\text{TJP},t)$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。

$TPM_{DRTM}(TJP)$	$\equiv w$ = read *m.pcr.d*; r = sign($PCR(s),w$),$AIK^{-1}(m)$; send r
Verifier(*TJP*)	$\equiv sig$ = recieve; v = verify *sig*, *AIK*(*m*); match v ,(*PCR*(*s*), *seq*(*vTPM Builder*(*TJP*),*vTPM-VM Binding*(*TJP*),*VM Builder*(*TJP*))

图 7 TVP-QT 中 m 信任传递的远程验证程序

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

$$\Gamma_{\mathrm{DRTM}} = \{\mathrm{Honest}(AIK(\mathrm{m}))),\quad \hat{V} \neq AIK(m)\}$$

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：

$$\Gamma_{\mathrm{DRTM}} \vdash [\mathrm{Verifier}(m)]_{V}^{t_B \cdot t_E}\ \exists t.(t<t_e) \wedge$$

$$(\mathrm{Mem}(m.pcr.d, seq(vTPM\ Builder(TJP), vTPM\text{-}VMBinding(TJP), VM\ Builder(TJP))@t) \quad (16)$$

$$\Gamma_{\mathrm{DRTM}},\ \mathrm{Protected}_{\mathrm{SRTM}}(m)$$

$$\vdash [\mathrm{Verifier}(m)]_{V}^{t_B \cdot t_E}\ \exists t.(t<t_E) \wedge$$

$$\mathrm{MeasureBoot}_{\mathrm{DRTM}}(m,t) \quad (17)$$

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 实例系统分析与讨论

为了在实际系统中检验 TVP-QT 及其信任链，课题组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；

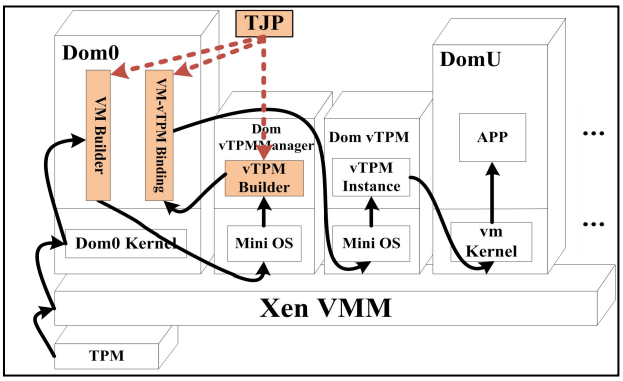

图 8 基于 Xen 的 TVP-QT 系统

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。

6 实验及结果分析

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：

表 1 物理平台(Dom0)和用户虚拟机(DomU-Ubuntu)配置

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。

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

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"]
.……
```

图 9 DomU-Ubuntu 配置部分参数

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"]
.……
```

图 10 Ubuntu vTPM 实例配置部分参数

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"]
……
```

图 11 vTPMManager 配置文件部分参数

6.1 可信衔接点 TJP 的本地验证及远程证明

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 PCR_i=Hash(Old PCR_i||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 2 所示。

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。

表 2 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 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	第一层：硬件 TPM 第二层：TCB
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	

```
PCR-00: 10 D8 FE 7C 6A E2 8B 72 2A D3 22 B3 13 B8 8E B3 E3 B8 31 9C
PCR-01: 3A 3F 78 8F 11 A4 B4 99 69 FC AA 8B CD 6E 39 57 C3 38 22 75
PCR-02: D3 9D 6B 7E CB 8E 30 22 AE A8 C9 25 84 68 40 28 3F 0B 47 20
PCR-03: 3A 3F 78 8F 11 A4 B4 99 69 FC AA 8B CD 6E 39 57 C3 38 22 75
PCR-04: 1B BA CD B9 EA 5B BC CD 3E 00 6B ED 7E 3F E0 03 02 40 DA 95
PCR-05: 2C 79 07 B3 03 C1 6D E4 DF 2B 27 EF C0 C1 A7 21 44 17 22 B1
PCR-06: 79 FA FA 72 54 71 FE 8A 87 1B C6 64 84 3F 36 3D B9 A7 B2 A3
PCR-07: 3A 3F 78 0F 11 A4 B4 99 69 FC AA 88 CD 6E 39 57 C3 3B 22 75
PCR-08: BF E1 E9 B4 BD 4E 61 47 1E 0F 3D 7E FA EB D4 41 02 06 3A 58
PCR-09: 8D AE DE E3 E5 F4 AD DE 0F C0 03 02 DA 3A 3F 34 E3 4A 32 45
PCR-10: CC BC EB B2 B4 C5 DB E3 0F AA 64 02 C1 36 0B 22 75 0B DA 0B
PCR-11: 85 86 59 B1 BE 86 FB 4C CD CB 11 3C A4 26 6E B9 45 44 94 66
PCR-12: 45 4D 0E E0 3D B6 03 D3 2E 21 7A E5 20 8D 3C 56 AF FB E6 91
PCR-13: 4F CC CF 7E F7 7A DB B8 4B 55 64 45 DE 6A F8 5A AA 7C 82 B1
PCR-14: 00 B5 BC B4 45 15 18 AD B3 45 DC DE 10 F4 BE 38 D5 2C C2 28
PCR-15: 9E 70 38 98 E9 DD BD EC CA E4 A3 B8 BC 6E B5 AD 98 BA E8 AA
PCR-16: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
PCR-17: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
PCR-18: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
PCR-19: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
PCR-20: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
PCR-21: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
PCR-22: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
PCR-23: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

图 12 信任链 PCR 信息

6.2 TVP-QT 性能测试及分析

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。

6.2.1 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。

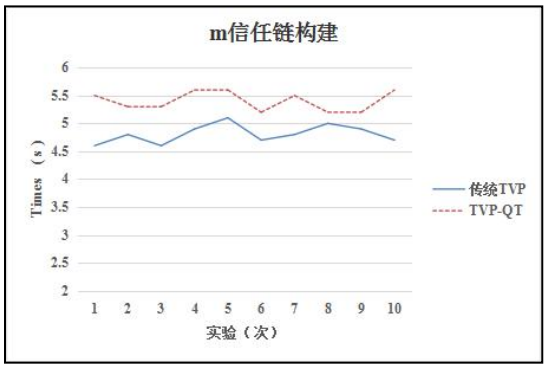

图 13　m 信任链构建时间

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。

6.2.2 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。

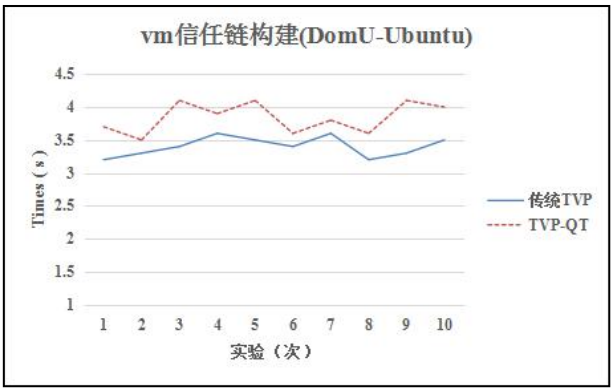

图 14　m 信任链构建时间

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。

7.结束语

利用可信计算技术构建可信虚拟平台并且构建信任链模型是目前解决云计算安全一个重要的研究方向。本文为了解决已有 TVP 模型过粗且逻辑不完全合理，而且还存在底层虚拟化平台和顶层用户虚拟机两条分离的信任链等问题，提出了一种具有可信衔接点的 TVP-QT 模型，并对 TVP-QT 中的功能组件及其信任属性进行详细定义，并结合可信

衔接点在 TVP-QT 建立从虚拟化平台硬件 TPM 开始的完整信任链模型 TVP-QT 信任链。最后，本文基于 Xen VMM 实现了 TVP-QT 原型系统，并对信任链 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。

参考文献

[1] 林闯,苏文博,孟坤,刘渠,刘卫东.云计算安全:架构、机制与模型评价[J].计算机学报,2013,09:1765-1784.

[2] 俞能海,郝卓,徐甲甲,张卫明,张驰.云安全研究进展综述[J]. 电子学报,2013,02:371-381.

[3] Ali M,Khan S U,Vasilakos A V.Security in cloud computing :opportunities and challenges[J].Information Science,2015,305:357-383.

[4] Xu P,Chen H,Zou D,et al.Fine-grained and heterogeneous proxy re-encryption for secure cloud storage[J].Chinese Science Bulletin,2014,59（32）:4201-4209.

[5] Xiang S,Zhao B,Yang A,et al.Dynamic measurement protocol in infrastructure as a service[J].Tsinghua Science and Technology,2014,19(5):470-477.

[6] Yu F,Zhang H,Zhao B,etal.A formal analysis of trusted platform module 2.0 hash based message authentication code authorization under digital rights management scenario[J].Security and Communication Networks,2015,8:2462 2476.

[7] 谭良,徐志伟. 基于可信计算平台的信任链传递研究进展[J]. 计算机科学,2008,10:15-18.

[8] 徐明迪,张焕国,张帆,杨连嘉. 可信系统信任链研究综述[J]. 电子学报,2014,10:2024-2031.

[9] 谭良,陈菊,周明天. 可信终端动态运行环境的可信证据收集机制[J]. 电子学报,2013,01:77 -85.

[10] 于爱民, 冯登国, 汪丹. 基于属性的远程证明模型[J]. 通信学报, 2010, 31(8):1-8.

[11] 谭良, 陈菊. 一种可信终端运行环境远程证明方案[J]. 软件学报, 2014(6):1273-1290.

[12] BERGER S, CACERES R, GOLDMAN K A, et al. VTPM: virtualiz-ing the trusted platform module[A]. Proc of the 15th USENIX Security Symposium[C]. Berkeley, USA, 2006. 305-320.

[13] CHRIS I D, DAVID P, WOLFGANG W, et al. Trusted virtual platforms: a key enabler for converged client devices[A]. Proc of the ACM SIGOPS Operating Systems Review[C]. New York, USA, 2009. 36-43.

[14] BERGER S, RAMON C, DIMITRIOS P, et al. TVDc:managing security in the trusted virtual datacenter[A]. Proc of ACM SIGOPS Operating Systems Review[C]. New York, USA, 2008. 40-47.

[15] KRAUTHEIM F J, DHANANJAV S P, ALAN T S. Introducing the trusted virtual environment module: a new mechanism for rooting trust in cloud computing[A]. Proc of the 3rd International Conference on Trust and Trustworthy Computing[C]. 2010.211-227.

[16] 王丽娜,高汉军,余荣威等.基于信任扩展的可信虚拟执行环境构建方法研究[J].通信学报, 2011, 32(9):1-8..

[17] T. GARFINKEL, B. PFAFF, J. CHOW, et al. 'Terra: a virtual machine-based platform for trusted computing", In Proc. Of SOSP'03, 2003.

[18] B. PFITZMANN, J. RIORDAN, C. STUBLE,et al. "The PERSEUS system architecture", Technical Report RZ 3335 (#93381), IBM Research Division, Zurich Laboratory, 2001.

[19] 常德显,冯登国,秦宇,张倩颖.基于扩展 LS^2 的可信虚拟平台信任链分析[J].通信学报,2013,34(5):31-41.

[20] Zhang Lei , Chen Xingshu, Liu Liang , Jin Xin.Trusted domain hierarchical model based on noninterference theory[J].The Journal of China Universities of Posts and Telecommunications .August 2015, 22(4): 7-16.

[21] Yu, Z., Zhang, W. & Dai, H. J A Trusted Architecture for Virtual Machines on Cloud Servers with Trusted Platform Module and Certificate Authority[J].Journal of Signal Processing Systems, 2017, Vol.86 (2-3), pp.327-336

[22] 池亚平,李欣,王艳,王慧丽. 基于 KVM 的可信虚拟化平台设计与实现[J]. 计算机工程与设计,2016,(06):1451-1455.

[23] 李海威,范博,李文锋. 一种可信虚拟平台构建方法的研究和改进[J]. 信息网络安全,2015,(01):1-5.

[24] 徐天琦,刘淑芬,韩璐. 基于 KVM 的可信虚拟化架构模型[J]. 吉林大学学报(理学版),2014,(03):531-534.

[25] 杨丽芳,刘琳. 基于虚拟机的可信计算安全平台架构设计[J]. 煤炭技术,2014,(02):170-172.

[26] 蔡谊,左晓栋. 面向虚拟化技术的可信计算平台研究[J]. 信息安全与通信保密,2013,(06):77-79.

[27] Scarlata, Vincent and Rozas, Carlos and Wiseman, Monty and Grawrock, David and Vishik, Claire: " TPM Virtualization: Building a General Framework"[C] , Trusted Computing(2007), 2007:43-56.

[28] F. John Krautheim*, Dhananjay S. Phatak, and Alan T. Sherman,Introducing the Trusted Virtual Environment Module:A New Mechanism for Rooting Trust in Cloud Computing[C],TRUST 2010, LNCS 6101, 2010:211 – 227.

[29] Shen Changxiang, Zhang Huanguo, Wang HuaiMin,etal. Research on trusted computing and its development [J].Science in China Series:E.2010,53(3):405-433.

[30] 朱智强. 混合云服务安全若干理论与关键技术研究[D].武汉大学,2011.

[31] 曲文涛. 虚拟机系统的可信检测与度量[D].上海交通大学,2010.

[32] GILLES B, GUSTAVO B, JUAN D C, et al. Formally verifying isolation and availability in an idealized model of virtualization[A]. Proc of the 17th International Conference on Formal Methods[C]. Berlin, Springer, 2011.231-245.

[33] DATTA A, FRANKLIN J, GARG D, et al. A logic of secure systems and its application to trusted computing[A]. Proc of the 30th IEEE Symposium on Security and Privacy[C]. Los Alamitos, USA, 2009. 221-236.

[34] Chen G,Jin H,Zou D,et al.Safestack:Automatically patching stack-based buffer overflow vulnerabilities[J].IEEE Transactions on Dependable and Secure Computing,2013,10(6):368-379

[35] Vermeulen S.SELinux Cookbook[M].Birmingha,UK:Packet Publishing Ltd,2014

[36] Varma P D,Radha V.Prevention of buffer overflow attacks using advanced stackguard[C]//Proc of 2010 International Conference on Advances in Communication,Network,and Computing,2010:357:359

[37] WANG Zhi, JIANG Xu- xian. HyperSafe:a lightweight approach to provide lifetime hypervisor control- flow integrity[C]/ /Proc of IEEE Sym-posium on Security and Privacy. Washington DC:IEEE Computer Society, 2010:380-395.

[38] JONATHAN M M, NING Q, LI Y L, et al. TrustVisor: efficient TCB reduction and attestation[A]. Proc of the IEEE Symposium on Security and Privacy[C]. Oakland, USA, 2010. 143-158.

[39] TAKEMURA C, CRAWFORD L. The Book of Xen: A Practical Guide for the System Administrator[M]. No Starch Press, 2009.

[40] Xen Project.[EB/OL] http://www.xenproject.org. 2017

[41] Mozilla Firefox Ltd.[EB/OL] http://www.firefox.com.cn/download/.

[42] Kingsoft Office Corporation.[EB/OL]http://linux.wps.cn/.2017

[43] CodeWeavers Inc.[EB/OL]https://www.winehq.org/.2017

[44] The Eclipse Foundation.[EB/OL]https://www.eclipse.org/downloads/.