OpenAnolis
龙 蜥 社 区

OpenAnolis Trusted Computing Technology Best Practices White-Paper

龙蜥社区可信计算技术

最佳实践白皮书

浪潮信息龙蜥联合实验室

版权声明

白皮书作者

（排名不分先后）

曹佩庆　陈洪博　陈　浩　陈　善

付月朋　韩春超　鲁　彬　龙　勤

李艺林　谭　琳　吴保锡　薛刚汝

许　鑫　应志伟　张　佳　张天佳

张　艺

目录

认识龙蜥

2020 年 9 月，依托多家头部厂商的多年技术沉淀，阿里云联合统信软件、Intel、Arm、龙芯中科、三大运营商等多家操作系统厂商、芯片公司、云计算公司共同发起龙蜥操作系统开源社区-OpenAnolis，立足云计算打造数字创新基石。作为面向国际的 Linux 服务器操作系统开源根社区及创新平台，龙蜥社区持续推动软、硬件及应用生态繁荣发展，到目前有超过 600 家合作伙伴参与共建。目前，龙蜥操作系统下载量已超 240 万，整体装机量达 500 多万，100 余款企业产品完成与龙蜥操作系统的适配。同时，浪潮信息、统信软件、中科方德、中国移动云、麒麟软件、中标软件、凝思软件、新支点、阿里云等 12 家企业基于龙蜥开源操作系统推出各自商业版本及产品，在政务、金融、交通、通信等领域累计服务用户超过 30 万。

龙蜥开源影响力

龙蜥社区及龙蜥操作系统也获得了一定的行业认可，工信部电子标准院首批开源项目成熟度评估，成为唯一获得“卓越级”认证的开源项目、龙蜥社区荣登 2022 科创中国”开源创新榜”、荣获“中国开源云联盟年度优秀开源项目奖”、“OSCAR 开源尖峰案例奖”等 25 项行业奖项。

● 龙蜥项目运作模式

龙蜥社区已成立 50+ 个 SIG 工作组，围绕芯片、内核、编译器、安全、虚拟化及云原生等操作系统核心领域进行技术创新，已发布龙蜥 Anolis OS 7、Anolis OS 8.x 系列、Anolis OS 23 公测版、Lifsea OS 等多个社区版本，为应对即将停服的 CentOS，官网已上线「CentOS 停服专区」为用户提供迁移方案及长期稳定支持，致力于成为 CentOS 的更佳替代。

● 龙蜥运营管理

“为更好地运营和治理社区，龙蜥社区定期召开月度运营委员会会议、技术委员会会议，理事大会。关于理事大会：龙蜥社区第二届理事大会圆满召开！理事换届选举、4 位特约顾问加入关于运营委员会会议：龙蜥社区第 15 次运营委员会会议顺利召开欢迎更多企业加入共建， 龙腾计划可参看：“龙腾计划”启动！邀请 500 家企业加入，与龙蜥社区一起拥抱无限生态”。

● 龙蜥开放的生态

为了鼓励合作伙伴在社区探索出更多的商业合作方式，真正牵引企业在龙蜥社区的合作落地，社区推出「龙腾计划」的升级版--「生态发展计划」，更聚焦在产品和商业合作本身。详情可参看：https://openanolis.cn/page/partner2

关于龙蜥操作系统 （ Anolis OS ）

龙蜥操作系统（Anolis OS）搭载了 ANCK 版本的内核，性能和稳定性经过历年“双 11”历练，能为云上典型用户场景带来 40% 的综合性能提升，故障率降低 50%，兼容 CentOS 生态，提供平滑的 CentOS 迁移方案，并提供全栈国密能力。最新的长期支持版本 Anolis OS 8.6 已发布，更多龙蜥自研，支持 X86_64 、RISC-V、Arm64、LoongArch 架构，完善适配 Intel、飞腾、海光、兆芯、鲲鹏、龙芯等主流芯片。

下载体验链接：*https://openanolis.cn/download*

2021 年 12 月 31 日，龙蜥开源社区（OpenAnolis）上线「CentOS 停服专区」，为受 CentOS 停服影响的用户提供迁移方案及长期稳定支持。此次停服，龙蜥操作系统（Anolis OS）产品**优势包括：打造系统化解决方案 AOMS、提供多款配套工具、承诺 10 年技术支持、兼容 CentOS 生态、具备差异化核心技术优势、历经丰富场景验证、沉淀用户迁移案例实践。**

反馈与共创

OpenAnolis 是一个开放包容的社区，因此我们也欢迎志同道合之士参与我们的文档修订。

对于文档中您认为不足之处，欢迎到我们的官方仓库 Whitebook TrustedComputing 新开 issue，我们会第一时间进行响应。

另外，若您想更新文档，也同样欢迎在 Whitebook TrustedComputing 提 PR。

实验室简介

浪潮信息龙蜥联合实验室以“平台共建、联合创新、繁荣生态”为目标，整合芯片、整机、板卡、数据库、中间件、ISV 等上下游生态力量，打造了一套完善的操作系统产业生态链，推动操作系统产业发展创新。实验室下设三个中心：

- **标准制定中心：**主要开展服务器操作系统相关标准制定工作，推动操作系统相关标准的落地与实施。
- **生态认证中心：**主要开展龙蜥及龙蜥衍生版硬件兼容性认证、适配认证体系建立等工作，不断完善操作系统生态体系。
- **联合创新中心：**主要面向智慧计算、云原生、系统安全等领域开展技术创新，联合生态伙伴进行课题研究与技术攻关，打造创新、开放、共享的实验室环境，持续推动技术创新发展。

实验室设施及业务概况

- **物理设施**：覆盖主流 x86、ARM 等架构的 100+服务器
- **支撑业务：**T-One、生态适配测试认证、先进技术联合探索
  - Anolis 服务器整机兼容性适配
  - 龙蜥衍生版操作系统 KOS 兼容性测试
  - 服务器部件兼容性测试
  - 数据库、中间件等软件兼容性测试
  - 系统安全联合研究及实践探索
- **参与厂商：**社区理事单位、社区及 KOS 生态厂商

实验室建设

- **商业：**商业衍生版发布、案例共建；人才与市场共建
- **技术**：SIG 贡献（Serverless SIG、可信计算 SIG）、技术贡献（特性、补丁、实践探索等）
- **产品生态：**产品认证、生态服务、运营支持
- **运营：**品牌站台、活动举办、资源支持

实验室动态：关注“浪潮信息操作系统”，及时获取联合实验室动态。

微信公众号

扫码关注微信公众号
“浪潮信息操作系统”

可信计算 SIG

SIG 成立背景

等保 2.0 时代通过可信计算技术构建关键信息基础设施主动安全纵深防御能力，已成为业界共识。龙蜥社区作为各大厂商的粘合剂，需要具备可信计算总体解决方案。各方也都希望有一个统一的可信计算技术解决方案，需要有一个社区驱动的参考实现，该实现需要具备良好易用性和可落地性。

当前国内可信计算技术生态面临如下困境：

1. 割裂的可信计算技术方案：
    a) 现有TPM、TCM、TDM、TPCM等可信计算技术落地应用是异构的、并不通用；
    b) 针对多元可信计算方案、可信计算软件栈其实是可以统一的；
    c) 当前开源软件栈已集成了国密算法、但稳定性和性能还需要进一步优化；
    d) PC 等终端领域客户比较期望有一个单机可用可感知的可信技术方案；
    e) 在数据中心领域远程证明更加关键；
    f) 龙蜥社区目前在 kernel、libgcrypt、openssl 等基础组件均已支持国密算法，但是基于 SM2/3/4 密码算法的远程证明全链路验证没有完全实现。
2. 没有统一可用的可信计算解决方案：

a) 客户不知道如何使用可信根提供的安全能力服务；

b) 作为可信计算技术落地应用的核心，远程证明 server 侧没有现成可用开源参考实践，TPM 虽然在 Windows 中非常成熟，但是在 LinuxOS 用户有感知地透传，应用可信计算技术目前比较有限；

c) 缺乏面向基础设施的通用可信计算特性使能开源参考实现。

为夯实关键信息基础设施主动安全纵深防御能力，解决当前国内可信计算技术应用面临的生态困境，促进可信计算技术开源开放生态繁荣发展，可信计算 SIG 由此成立。

后续，SIG 将依托浪潮信息龙蜥联合实验室丰富多样的服务器可信计算平台设备环境，联合 SIG 成员伙伴共同推动可信计算实践及解决方案探索，输出易用好用的可信计算解决方案及配套成果，繁荣国内可信计算开源开放生态。

SIG 愿景

- 立足龙蜥社区国内 OS 根社区定位，将龙蜥社区可信计算 SIG 打造为国内可信计算技术开源开放的总入口；
- 依托社区及 SIG 成员单位携手挖掘易用实用的可信计算技术落地方案，让用户清晰感知可信计算价值，让用户感知可信计算价，助力可信计算应用推广；
- 让 SIG 成为国内可信计算技术应用推广的践行者。

SIG 目标

- **开源可信基础软件开发维护：**面向异构可信计算技术方案，提供通用的可信计算基础软件实现（包括可信软件栈、远程证明组件等），实现可信基础软件全栈国密支持，持续优化可信基础软件的稳定性和易用性；
- **探索可信计算落地方案：** 面向不同场景（如云服务、数据中心、桌面端、边缘侧等），探索可落地的可信计算解决方案参考实现，提供相应的代码仓库和发行版；
- **可信计算技术动态：** 发布年度可信计算技术白皮书。

可信计算白皮书（年度）	**可信计算整体解决方案**	• 国内外可信计算开源成果 • 国内外可信计算标准 • 龙蜥OS可信计算成果使用指导 • 可信计算落地方案探索成果	
落地方案探索	**龙蜥OS最佳实践**	轻量级远程认证方案、轻量级可信根部署工具、可信网络连接（TCA/TNC）、基于机密计算的可信计算实践、基于可信根的全盘加密安全增强、基于可信连接的SSH访问安全增强、基于可信根的PKCS服务.....	
可信基础软件开发与维护	**可信根服务引擎**	RoT-TSS-engine	全栈国密支持
	远程证明	Remote attenstation	
	可信根工具集	RoT tools、模拟可信根、测试套件等	
	可信软件栈/可信软件基	TSS/TSB	
	内核可信特性	tddl、IMA等	

加入可信计算 SIG

以下是我们在 OpenAnolis 上的可信计算 SIG，非常欢迎有兴趣的开发能参与到社区中来，携手繁荣中国可信计算开源生态、推动可信计算高速发展。

🔗 SIG 地址：*https://openanolis.cn/sig/tc-sig*

欢迎加入钉钉或微信交流群，与社区用户和开发者实时交流：

- 钉钉群：“龙蜥-可信计算 SIG 技术交流群”，群号：15370024496

- 微信群：“龙蜥-可信计算 SIG 技术交流群”

微信公众号

扫码关注微信公众号
“OpenAnolis龙蜥”

可信计算SIG技术交流群

扫码加入钉钉群
交流可信计算热门技术

可信计算SIG技术交流群

扫码加入微信群
交流可信计算热门技术

1. 可信计算概述

可信计算是通过检测和强化实体行为的预期性来保障实体信任的技术，是一种从体系结构着手解决信息系统安全问题的技术理念；其基本思想是先在计算机系统中建立一个信任根（基），信任根的可信性由物理安全、技术安全与管理安全共同确保；再建立一条信任链，从信任根开始，到硬件平台，到操作系统，再到应用，一级度量认证一级，一级信任一级，把这种信任扩展到整个计算机系统。 可信并不等同于安全，但可信是安全的基础，因为安全组件、策略只有运行在可信的环境下才能进一步达到安全目的。通过系统和安全组件的完整性保障，可以确保业务应用使用正确的软件栈，并在软件栈受到攻击发生改变后能及时发现。总之，在系统和应用中引入可信计算能够极大地降低由于使用未知或遭到篡改的系统/软件遭到攻击的可能性。

本节主要介绍可信计算技术的应用及发展现状、可信计算关键技术与术语以及可信计算技术当前面临的挑战与未来趋势等三部分内容，便于读者理解白皮书内容。

1.1 可信计算应用发展现状

可信计算概念最早可追溯到 1983 美国国防部的 TCSEC 准则及之后出现的彩虹系列信息系统安全文件。1999 年，IBM、微软、Intel 等企业成了 TCPA（2003 年改名为 TCG），主要致力于形成可信计算的工业标准。目前 TCG 已经制定了包括 TPM、TSS、TNC 等一系列技术规范，并形成了针对 IoT、云计算、个人终端、移动终端、存储等不同场景的工作组，致力于可信计算在相关场景的应用标准编制与解决方案推动。

国际上已经形成以 Trusted Computing Group（TCG）为代表的可信计算组织，推动 TPM 规范在 PC、服务器、移动终端、网络、云计算、物联网等领域的应用。在服务器及云计算领域，国际 IT 巨头已将可信计算技术作为其产品的重要支撑。Intel 服务器 CPU 已经全面支持 TPM2.0；微软 Windows Server 2012+ 已支持 TPM2.0，支持可信云计算环境的构建；Linux Kernel4.0 已经集成 TPM2.0，以及主流虚拟化软件 Xen、KVM、Openstack、VMware 等都提供了对 TPM 和 vTPM 的支持；IBM 收购的 Softlayer 公司为全球 60 多个重要客户提供可信云主机服务；同时，许多芯片公司都将部分可信计算功能集成到商用的处理器中，如 ARM 的 TrustZone 技术、 Intel 的 SGX 技术和 AMD 公司的 SEV（secure encrypted virtualization）技术等，都在处理器中实现了内存隔离，可以为上层应用提供安全的执行环境，保障敏感程序的安全性，并被广泛应用在移动手机和云平台中。

鉴于可信计算技术对国家信息安全体系的重要性，经国家密码管理局批准，中国于 2006 年成立了可信计算密码专项组，并于 2008 年 12 月更名为中国可信计算工作组(China TCM Union)，简称 TCMU。2007 年 12 月，国家密码管理局颁布了《可

信计算密码支撑平台功能与接口规范》，将国内使用的可信基础模块定义为TCM(trust cryptography module)。相较于 TPM, TCM 采用了我国《商用密码管理条例》中规定的 SM2、SM3 等国密算法, 同时引入了对称密钥算法, 简化了 TPM 中复杂的密钥管理。TCM 的证书认证机制采用签名密钥以及加密密钥的双证书机制, 将对称密钥与非对称密钥结合保护系统安全, 在密钥管理体系和基础密码服务体系等方面进行了改进, 提升了系统的安全性。 TPM 和 TCM 的构成和功能类似, 提供可信计算平台的信任根(RTS, RTR), 是由 CPU、存储器、I/O、密码协处理器、随机数产生器和嵌入式操作系统等部件组成的独立 SoC 芯片, 具备可信度量的存储、可信度量的报告、密钥产生、加密和签名、数据安全存储等功能。2015 年 TPM 2.0（Trusted Platform Module）library specification 正式成为国际标准 ISO/IEC 11889，吸纳了 TCM 中相关的安全改进，并首次成体系支持中国密码算法体系，包括 SM2/SM3/SM4 密码算法。这是中国密码算法技术和标准的又一次重要突破，也是中国信息安全标准在国际标准化工作中的重要进展。ISO/IEC 11889 支持中国商用密码算法体系（SM2/SM3/SM4），使得在数据安全保护上更加牢不可破[1]。

我国学者则从传统计算机体系结构着手，考虑到传统冯诺伊曼架构存在的安全设计缺陷，提出了“在计算运行的同时进行安全防护的可信计算模式，以密码为基因产生抗体，实施身份识别、状态度量、保密存储等功能，及时识别自己和非自己成份，从而破坏和排斥进入机体的有害物质”即“主动免疫安全可信计算”技术体系（这一体系被学界称为“可信计算 3.0”）。主动免疫安全可信计算通过构建“计算+防护”并行双体系结构，实现防护部件并行动态的方式对计算部件运算过错进行可信验证，达到主动免疫防护效果。在并行的双体系结构中，采用了一种安全可信策

略管控下的运算和防护并列的主动免疫的新计算体系结构，以可信密码模块（TCM）连接可信平台控制模块（TPCM），组成可信根，由策略产生可信验证规则，由可信软件基根据安全可信策略规则实施身份识别、状态度量、保密存储等功能，及时发现异常并加以处置，从根本上防止（恶意代码）对计算部件（主机）的攻击。该体系的可信验证通过对人的操作访问策略 4 要素（主体、客体、操作、环境）进行动态可信度量、识别和控制，以达到纠正传统访问控制策略模型局限于授权标识属性进行操作，而不作可信验证、难防篡改的安全缺陷。另外，传统访问控制不考虑环境要素（代码及参数）是否被破坏，难以防止恶意代码攻击，为此必须对环境要素进行可信验证的基础上依据策略规则进行动态访问控制[2]。当前，国内主要以中关村可信计算产业联盟代表推动可信计算 3.0 技术体系及其生态的繁荣发展,推出了一系列团体标准，并通过“等级保护 2.0 与可信计算 3.0 联合攻关基地”推动相关团体标准的落地实施。

1.2 可信计算关键技术及术语

1.2.1 信任根概述

信任根是可信计算机系统可信的基础，可信计算平台包含三个信任根：可信度量根(Root of Trust for Measurement, RTM)、可信报告根(Root of Trust for Report, RTR)、可信存储根(Root of Trust for Storage, RTS)。如下图所示：

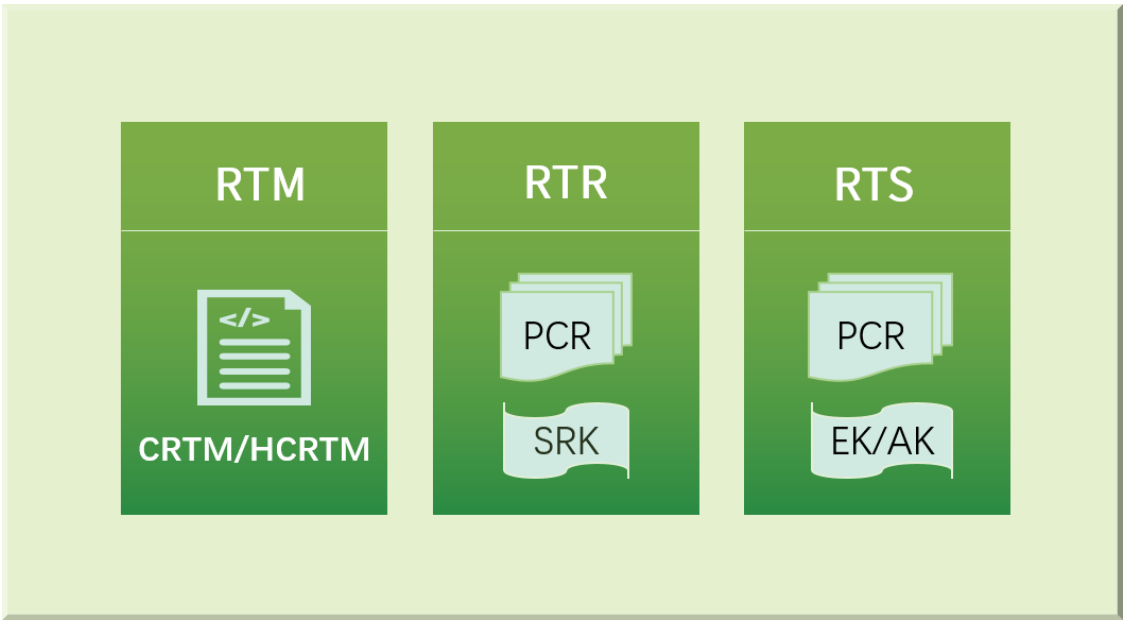


图 1-2-1 可信计算平台信任根构成

度量存储报告机制是可信计算平台重要工作机制，这一机制是指“对可信计算平台进行度量并将度量产生的可信度量值进行存储，当访问客体询问时提供可信报告”，这一机制是确保可信计算平台自身可信并向外提供可信服务的基础。RTM 是对可信计算平台进行度量的信任基点，它的本质是一段代码；RTS 是平台可度量值可信存储的基点，它是由信任根（如 TPM/TCM）中的平台状态寄存器（PCR）和信任根中的存储根密钥（SRK）共同构成；RTR 是平台向访问客体提供可信报告的信任基点，它是由 PCR 和信任根中的背书密钥（EK）共同构成。它们（RTR/RTM/RTS）共同构成了可信计算平台的信任基础。

信任根的可信性由物理、技术和管理等方面的安全措施共同确保。

常见可信根及其形态 目前国内外比较成熟的可信根规范有 TPM（可信平台模块）、TCM（可信密码模块）及 TPCM（可信平台控制模块）。如下表所示：

表 1–2-1 常见可信根及其形态

类型	成熟度	标准规范	技术路线	实现形态
TPM	具备成熟的产业生态和健全开源开放体系	TCG TPM library 系列规范，当前主要应用的 TPM2.0 规范定义的可信根	被动可信：可信根由处理器拉起	● 独立芯片实现，接口主要为 SPI 或 I2C ● 在处理器可信执行环境运行（即 fTPM），多见于 ARM 平台 ● 软件模拟（即 swtpm），主要用于研究开发与虚拟化场景
TCM	TCM1.0 在服务器领域应用较少，开源成果有限，随着 TCM2.0 规范的发布，与国际可信计算开源生态的融合，未来 TCM 在 PC Client 及服务器领域的应用将丰富起来。	国家密码局牵头发布的系列 TCM 规范： 当前已发布 TCM2.0， TCM2.0 配套的软件服务模块（TSM）标准正在修订中	被动可信：可信根由处理器拉起	● 独立芯片实现，接口主要为 SPI 或 I2C ● 软件模拟，主要用于研究开发与虚拟化场景 ● 处理器内置，提高一个独立的核作为 TCM 的计算资源
TPCM	TPCM 由国内学者提出，用于实现可信计算 3.0 平台构建。在特种领域应用成熟、开源生态方面的建设亟待提升。	已发布 TPCM 国家标准及配套的可信软件基（TSB）国家标准。	主动可信：可信根优先上电运行且相对独立	● 独立版卡，接口多为 PCIe、也有为 USB 接口 ● 软件模拟，主要用于研究开发与虚拟化场景 ● 处理器内置，处理器开辟出一个专用的核作为 TPCM 的

类型	成熟度	标准规范	技术路线	实现形态
				计算单元，同时提供专用的计算存储资源

1.2.2 可信度量与信任链

可信计算中采用对客体哈希的方式实现客体完整性值的采集，通过实时采集到的完整性值与客体完整性基准值比对的方式验证客体的可信状态。对客体进行哈希、将哈希结果扩展到可信根、并记录事件日志的过程称为可信度量或度量事件。 如下图所示。

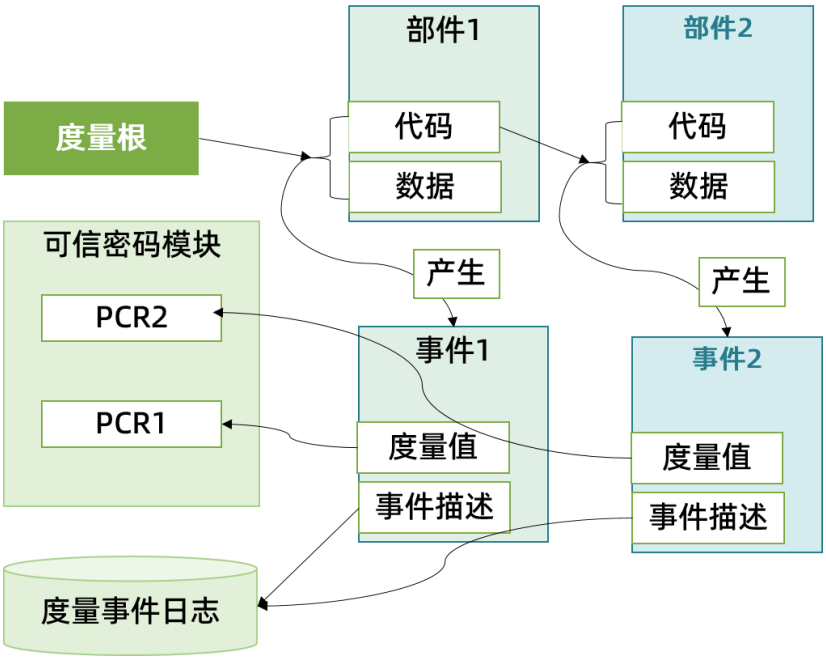


图 1-2-2 可信度量事件示意图

信任链的主要作用是将信任关系扩展到整个计算机平台，它建立在信任根的基础上。信任链可以通过可信度量机制来获取各种各样影响平台可信性的数据，并通过将这些数据与预期数据进行比较，来判断平台的可信性[3]。

建立信任链时遵循以下三条规则：

1) 所有模块或组件，除了 CRTM（信任链构建起点，第一段运行的用于可信度量的代码），在没有经过度量以前，均认为是不可信的。同时，只有通过可信度量并且与预期数据相符的模块或组件，才可归入可信边界内。

2) 可信边界内部的模块或组件，可以作为验证代理，对尚未完成验证的模块或组件进行完整性验证。

3) 只有可信边界内的模块或组件，才可以获得相关的 TPM/TCM 控制权，可信边界以外的模块或组件无法控制或使用可信平台模块

下图所示为基于 TPM2.0/TCM2.0 的信任链构建示意图。

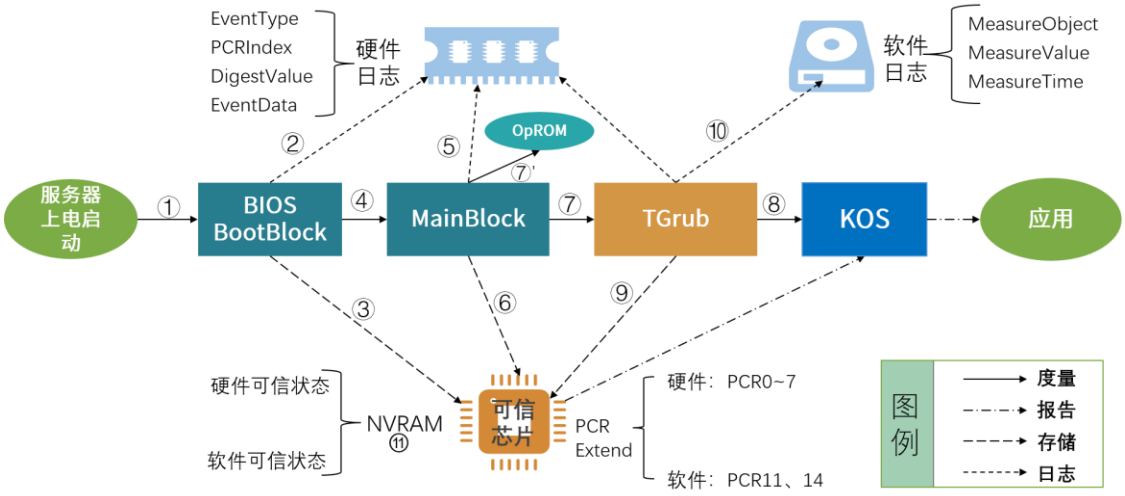


图 1-2-3 可信计算平台信任链

1.2.3 可信计算平台

可信计算平台是指具有可信计算安全机制并能够提供可信服务的计算平台.其主要特征是：有信任根，并基于此构建信任链机制、具有度量存储报告机制、能够提

供确保系统数据完整性、数据安全存储和平台远程证明等可信功能服务[4]。典型的可信计算平台包括:可信 PC、可信服务器、可信 PDA 等。

可信计算平台以 CRTM（可信度量根核）为起点，以信任链的方式来度量整个平台资源的完整性，将度量值扩展到可信根的平台配置寄存器中，并通过可信芯片向询问平台可信状态的实体提供报告，供访问者判断平台是否可信，以决定是否可以与其交互。这种工作机制被称为可信度量、可信报告和可信存储机制，是可信计算机和普通计算机在安全机制上的最大区别。

1.2.4 可信软件栈

可信软件栈是为了方便操作系统层面的安全应用访问可信芯片提供的服务，从而实现为用户提供可信服务。

- 位于用户应用软件与可信芯片之间，作为使用可信芯片的入口。
- 提供安全芯片访问、安全认证、密码服务和资源管理等功能。
- 为应用程序提供一套函数接口来同步访问可信芯片的功能 。
- 对安全芯片自身有限的资源进行管理；管理多个应用程序访问可信芯片资源的请求，提供可信全芯片的并发访问。
- 以合理的字节流顺序及赋值对上层应用隐藏内部命令处理过程 。
- 解决可信芯片自身接口的复杂性和对外服务的不便性 。
- TPM 对应可信软件栈 TSS（TCG Software Stack）。
- TCM 对应 TCM 服务模块 TSM（TCM Service Module）。

1.2.5 远程证明

远程证明是指可信计算平台向外部实体证明平台身份及可信状态的过程。相对于基于身份的认证机制，远程证明进一步扩展和丰富了认证的内容，使得认证的实体能够对认证客体进行更深层次更细粒度的认证。有效的避免了基于身份认证的过程中对实体安全状态一无所知的窘境。远程证明不仅对用户和平台身份进行了认证、还进一步的对平台的安全状态、软硬件配置等信息进行了验证，保证了平台的状态符合预期的安全策略，从源头上消除了大量潜在的安全攻击。

基于 TPM/TCM 的远程证明与上述目标一致，是一种验证计算机系统软硬件配置正确性的方法。在这个方法中，验证者向系统发送挑战查询，选择要验证的范围。证明者收集描述系统的引导状态、当前配置和身份的证据。背书密钥用于签署证明证据，其公钥会传递给验证者。验证者使用密码学方式检查证据是否满足其信任评估策略，并确定设备是否来自期望的制造商。

远程证明基本流程： 完整的远程证明流程包括两个主要步骤：

1) 服务器验证客户端可信根设备

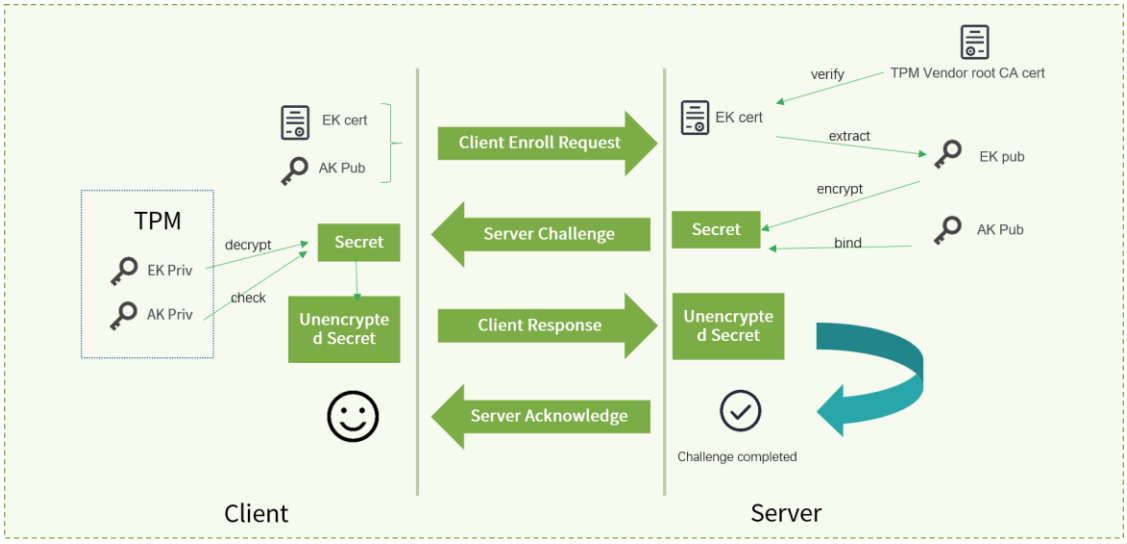


图 1-2-4 远程证明示意图 1

每个可信根（TPM/TCM）设备都有一个唯一的非对称密钥，称为背书密钥（Endorsement Key，EK）。其中公钥被称为 EKPub，私钥被称为 EKPriv。EK 由 TPM 制造商在生产过程中烧录到 TPM 芯片中，并由制造商颁发 EKCert 作为公钥的数字签名。

- 为了验证可信根设备的真实性，需要建立对 EKPub 或 EKCert 的信任。这通常是通过与颁发 EKCert 的证书颁发机构（CA）进行交互来实现的。
- 用户会生成一个密钥对 AK，并向 Server 证明该密钥与可信根的 EKPub 存在加密关系，并且用户拥有 EKPriv。
- AK 是在可信根内部由 EK 派生的一个密钥对，用于生成远程证明的可信报告。每次创建 AK 都会生成一个全新的密钥，使得该密钥具备匿名性。
- 用户向 Server 提交证书请求，并将自己的 AKPub 和与可信根的 EKPub 相关的证明信息提交给 Server。

- Server 对用户提供的信息进行验证，以确保用户的密钥 AK 与可信根的 EKPub 存在加密关系，并且用户具有对应的私钥。
- 如果验证成功，Server 会向用户颁发一个证书，表示用户的密钥已通过由可信根保护的证明过程。

2) 服务器验证来自客户端的度量事件日志

度量事件日志完整性验证是指确保存储在可信根（如 TPM）中的事件日志未被篡改且保持可信的过程。

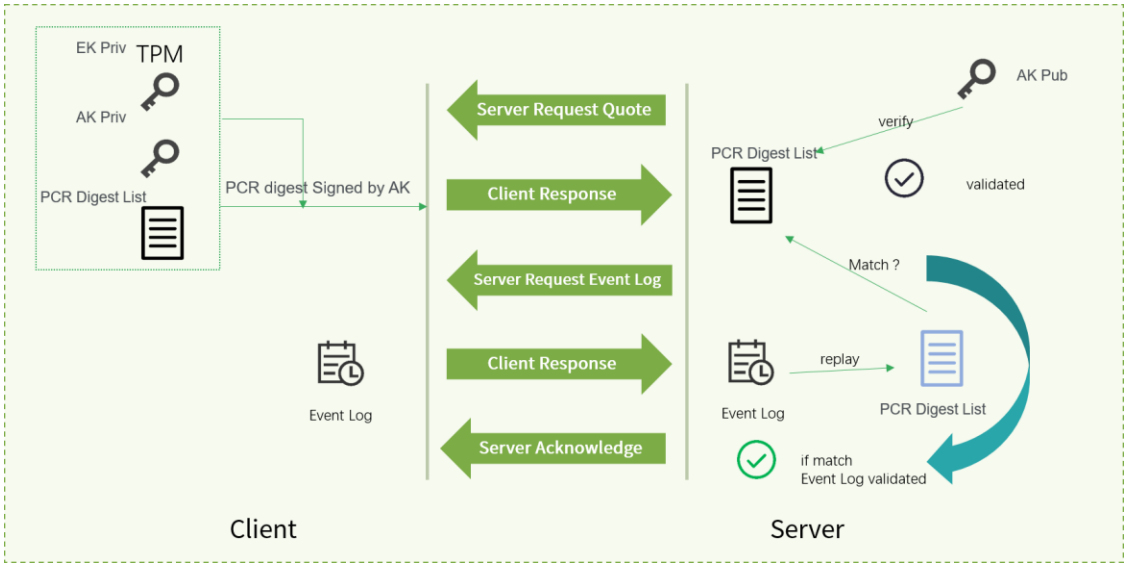


图 1-2-5 远程证明示意图 2

在引导过程中，可信根会记录一系列的事件，例如启动固件的验证、启动操作系统的过程等。这些事件会被存储在 TPM 的事件日志中，以提供关于系统引导过程的可信证据。可信根会使用称为“度量”（measure）的过程来记录引导过程中的各个阶段的度量值。这些度量值是通过对引导代码和配置进行哈希计算得到的。这些度量值会被存储在事件日志中，并且使用可信根的密钥进行签名，以确保其完整性和真实性。

- 服务器向客户端平台请求 quote。
- 客户端平台使用 AK 私钥将 PCR 列表作为 quote 进行签名，并将 quote 发送给服务器。
- 服务器使用 AKPub 验证 quote 的签名。如果验证成功，则服务器确认 PCR 列表是真实的。一旦验证成功，服务器会发送请求以获取事件日志。
- 客户端平台直接将事件日志发送给服务器。
- 服务器回放事件日志以还原 PCR 值。如果还原的 PCR 值与原始 PCR 值相同，则服务器确认事件日志是真实的。

1.2.6 可信网络连接

TNC 可信连接架构

可信网络连接（TNC）是对可信平台应用的扩展，其目的是是确保网络访问者的完整性，其认证过程可简述为：终端接入网络的过程中先后对终端用户身份、终端平台身份、终端平台可信状态等信息进行认证，只有全部满足认证需求的情况下才允许接入网络。

TNC 的架构分为 3 层：

- 网络访问层: 从属于传统的网络互联和安全层, 支持现有的如 VPN 和 802.1X 等技术, 这一层包括 NAR (网络访问请求)、PEP (策略执行) 和 NAA (网络访问管理) 3 个组件；
- 完整性评估层:这一层依据一定的安全策略评估 AR(访问请求者)的完整性状况；

- 完整性度量层: 这一层负责搜集和验证 AR 的完整性信息.TNC 通过网络访问请求, 搜集和验证请求者的完整性信息, 依据一定的安全策略对这些信息进行评估, 决定是否允许请求者与网络连接, 从而确保网络连接的可信性。

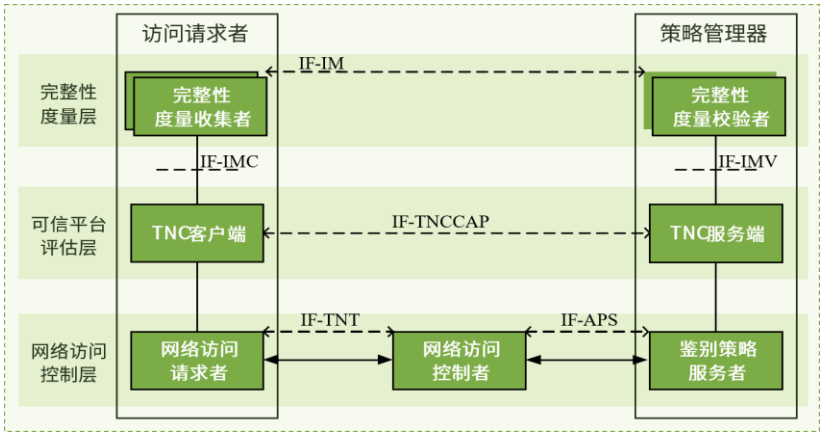


图 1-2-6 TNC 协议架构

TCA 可信连接架构

TNC 本质上其实是一个二元架构，这里只涉及访问请求者和网络接入方，且网络接入方处于控制地位，整个网络接入过程的可信认证都是为了确保网络的安全，却无法确保终端安全，即终端无法确认所接入的网络是否可信，从而无法确保获取的网络服务是否可信。 我国于 2005 年开始制定的可信连接架构（TCA），有效的解决了该问题。TCA 在制定过程中参考了 TCG 的 TNC 标准，同时采用了具有自主知识产权的三元对等实体鉴别及访问控制方法。

TCA 架构如下图所示，由图可知该架构设计包括可三个实体、三个层次和若干组件。相比于 TNC，该架构最大特点是采用了三元对等架构，将访问请求者和平台控制器视为对等实体，其中策略管理器作为可信第三方，提供访问请求者和访问控

制者之间的双向身份认证。这种方式既简化了身份管理、策略管理和证书管理机制，又保证了终端与网络的双向认证。

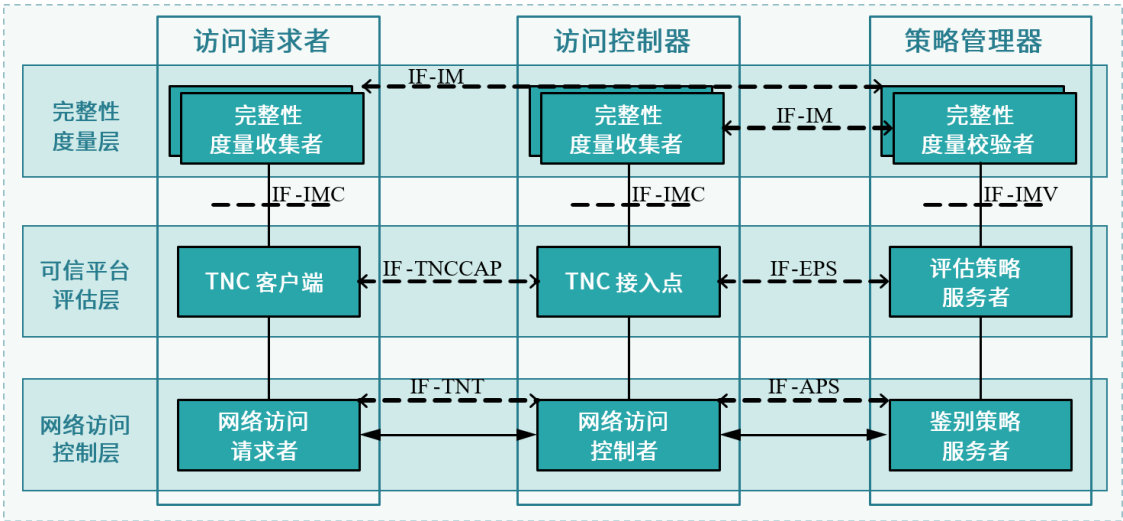


图 1-2-7 TCA 协议架构

1.2.7 TCM2.0/TPM2.0 特性概述

TCM2.0/TPM2.0 是本白皮书主体内容重点介绍了基于 TPM2.0/TCM2.0 为可信根的的可信计算最佳实践，因此本节简要介绍 TCM2.0/TPM2.0 的关键特性，便于读者理解白皮书主体内容。

本节的撰写参考了威尔·亚瑟等关于 TPM 的原理及实践指南的著作[5]。

注：为了叙述方便，在没有明确说明的情况下本节下文中“可信根”指代 TPM/TCM。

TCM2.0/TPM2.0 提供的安全功能：

- 设备识别：在发布没有可信根规范之前，设备主要通过 MAC 地址或 IP 地址来识别，而不是靠安全标识符。
- 密钥安全生成：创建密钥时，提供基于硬件的随机数生成器。

- 密安全存储：基于可信根可以在设备遭到软件攻击时为设备密钥提供芯片级安全保障。
- NVRAM 存储：当 IT 组织获取新设备时，它经常擦除硬盘并以组织的标准加载方式重写磁盘，而具有 NVRAM 则允许 TPM 能维护证书存储。
- 设备健康证明：在拥有 TPM 系统之前，IT 组织使用软件来证明系统的健康状况但是如果一个系统受到破坏时，软件可能会在系统不健康的时候也报告它是健康的。
- 算法灵活性：如果算法被证明在加载时符合预期，则可以在不重新修订规范的情况下进行更改。
- 增强授权：这种新功能统一了可信根中所有实体的授权方式，同时扩展了 TPM 启用考虑多因素和多用户身份验证的授权策略的能力，以及额外的管理功能。
- 非脆性 PCR：上一代可信根将密钥与设备状态锁定时会导致管理问题，因为，在通常情况下，当设备状态必须由授权状态更改时，密钥也必须更改。现在不再是这样。
- 灵活管理：不同种类的授权可以分开，从而允许更灵活地管理 TPM 资源。
- 按名称识别资源：上一代可信根设计中的间接引用带来了安全挑战。而这些问题已经通过为所有的可信根资源使用加密的安全名称修复了。

2. 关键技术机制及功能特性

(1) 哈希扩展

“扩展”并不是常见的密码学用语，但是这个操作却贯穿了整个可信根。扩展操作包含以下步骤：

1) 存在一个值 A；

2) A 与另一个用于扩展的值 B 进行连接，得到消息，A‖B；

3) 对得到的消息进行哈希，生成 hash(A‖B)；

4) 用这个哈希结果替换原来的 A。

整个过程可总结为：A<-hash(原始 A‖B)。 例如用 SHA-1 对消息“abc”的摘要，用全 0 进行扩展，得到: “ccd5bd41458de644ac34a2478b58ff819 bef5acf”扩展值是由一系列的扩展操作生成的，它有效地记录了消息扩展的过程。由于安全哈希的特性，我们无法回溯得到之前的值。因此，一旦消息被扩展，就不能通过逆运算取消操作或删除历史记录。然而，以 32 字节的 SHA-256 算法为例，值的实际长度与消息的护展次数无关，这个固定长度使得它可以进行无限次的记录，而这个特性在内存有限的 TPM 中是至关重要的。 扩展机制用于更新平台配置寄存器(PCR)的值。PCR 是一个用于存放哈希值的寄存器扩展人 PCR 中的值能代表平台状态。假设 PCR 值表示当前平台不可信，但攻击者想将其复改成可信，那么他就会构造另一个消息，这个消息需以当前的 PCR 值开头，这样才能得到一个表示可信的哈希值。但由于数值已经更新，安全哈希的不可逆特性使得这个攻击并不能实现。 扩展也运用于 TPM 的审计日志中。审计日志记录了 TPM 的请求及回复信息。由于扩展特性，一个审计项一旦被添加到日志后就不能移除，且日志的大小与审计过的请求数量无关。 扩展在创建认证策略时也会用到。

（2）PCR（平台配置寄存器）

平台配置寄存器(PCR)是可信根的关键特性之一。它的主要用途是提供一种密码记录(度量)软件状态的方法，包括平台上运行的软件和该软件使用的配置数据。PCR 更新计算被称为“扩展”，是一种单向哈希计算(PCR 新值=HASH(PCR 旧值||扩展的数据))。这样的计算结果将无法被破解。然后人们可取这些 PCR 的值来报告它们的状态。PCR 也可以通过签名来返回一份更安全的报告称为验证(attestation)或 quote(引用证实)。人们也可以在增强授权策略中使用 PCR 来限制其它对象的使用。可信根从不对度量结果进行判断。在内部，它不知道这些度量值是好还是坏，不知道者些度量值是否安全可信。在度量时，TPM PCR 只记录值。当应用程序要在授权策略用 PCR 值，或者远程方通过询问 PCR 值的签名验证并判断其可信性时，将随之形成安信任关系。 可信根包含多个 PCR。PCR 依照约定分配到各个软从启动代码到操作系统再到应用程序。同时，PCR 也被分配给要运行的软件(通常分配偶数编号的 PCR)和定制引导过程的配置文件(通常分配奇数编号的 PCR)。

（3）NVRAM 存储

在 PC 中带有限制访问控制属性的少量 NVRAM 存储器是非常有用的。它可以存储密钥，一旦 PC 关闭，这些密钥便不可用，这比使用公钥/私钥对的解密方法能更快地访问数据，并且提供在系统的不同部分之间来回传递消息的机制。TPM 中的 NVRAM 能被配置成分别控制读写功能，这意味着可以向用户提供一些数据，而不用担心由于意外或恶意的意图而被擦除。此外，你可以使用 NVRAM 存储当 PC 无法访问其主存储时使用的密钥。PC 无法访问其主存的情况可能发生在引导周期早期或者在自加密驱动被赋予密码之前，从而允许读取 NVRAM 中存储的密钥。

（4）实体（entity）

实体(entity)是可信根中一个可以通过句柄直接索引到的项目，可信根实体类型包括：

- 永久性实体(hierarchy、字典攻击锁定机制和 PCR)；
- 非易失性实体(NV 索引)；
- 对象(密钥和数据)；
- 易失性实体(各种类型的会话)。

（5）层级（Hierarchy）

hierarchy 是一个实体集合，其相互关联，并作为一个组来被管理。这些实体包括永久性实体(hierarchy 可柄)在树根的主对象和其他对象，如树中的密钥。NV 索引属于一个 hierarchy，但不在树中。除永久性实体外，实体可以作为一个组被删除。 每个 herarchy 的密码根都是一个种子（一个大的随机数），由可信根生成，永远不会泄露到它的安全边界之外。可信根用这个种子创建主对象，如存储根密钥。这些在 hierarchy 顶部的密钥就是父亲，用来加密它们的孩子。 每个 hierarchy 有一个相应的 proof 值。proof 可以独立生成，也可基于 hierarchy 种子生成。可信根使用 proof 值，确保提供给可信根的值是由该可信根初始生成的。通常，可信根基于 proof 生成一个 HMAC 密钥，然后可信根在其内部自己生成一段 HMAC 数据。后面在有数据响应给 TPM 时，通过检查 HMAC 判断数据的真实性。hierarchy 可以持久存在(在重启时被保存)，也可以是易失的(重启时被擦除)。每个 hierarchy 都针对具体的用例:针对平台制造商、针对用户、针对隐私敏感的应用程序和临时需求。可信根中的 Hirerarchy 的类型包括：

- 持久性 Hierarchy：包括平台 Hierarchy（供平台制造商使用）、存储 Hierarchy（供平台所有者使用）和背书 Hierarchy（提供用户隐私保障，整体是一个隐私相关密钥树，由隐私管理者控制）；
- 空(NULL)Hierarchy：空 hierarchy 与三种持久性 hierarchy 类似。它可以有主密钥，基于主密钥可以创建子密钥。当时用空 Hierarchy 时、可以把可信根当做一个密码协处理器看待。需要说明的是空 Hierarchy 的种子不是持久的，可信根每次重启后生成的种子是不一致的。

（6）增强授权

在可信根中将对实体使用限制的综合称为策略、用于实现在特定场景下使用实体的限制。增强授权(EA)主要使用以下类型的授权：

- 密码（明文）： 在某些环境中，例如在操作系统启动之前 BIOS 具有可信根的控制权的情况下，通过使用 HMAC 获得的附加安全性不保证利用 HMAC 授权使用可信根服务的额外软件成本和复杂性。
- HMAC 密钥：在某些情况下，特别是当用作 TPM 通信接口的操作系统不被信任，但是与 TPM 通信的软件是可信时，使用 HMAC 授权增加的成本和复杂性是有必要的，例如在远程系统上使用 TPM。
- 签名（例如，通过智能卡）：智能卡主要是在运维时防止滥用 IT 组织权限。当员工离岗时，可以回收智能卡，它不会像密码那样容易暴露。
- 附加数据签名：例如，额外的数据可以是通过特定的指纹识别器识别的指纹这是 EA 中特有的新功能。例如，生物特征读取器可以识别已经匹配生物特

征的特定的人或者 GPS 可以确认机器处于特定区域。这避免了可信根必须匹配指纹或了解 GPS 坐标的含义。

- 启动时 PCR 值作为系统状态的代理：如果系统管理模块软件已经被破坏，则可以防止释放全磁盘加密密钥。
- 位置作为特定命令来源地点的代理：到目前为止，这仅用于指示某个命令是否来自于 CPU 对特殊请求的响应，例如基于 IntelTXT 和 AMD 的 AMD-v 实现的 Flicker（已开源，由卡内基·梅隆大学贡献），使用它可提供一个小型、安全的操作系统。这个操作系统可以在需要执行安全操作时触发。
- 时间：制定相关的策略将密钥的使用限制在某些时间内。这就像银行的时间锁，只允许在营业时间内打开保险柜。
- 内部计数器值：仅当内部计数器处于某些值之间时，才能使用对象。这种方法有助于设置只能使用一定次数的密钥。
- NV 索引值：一个密钥的使用被限制在某些位设置为 1 或 0 时。这对撤销访问密钥很有用。
- NV 索引：可以基于 NV 索引是否已写入来进行授权。
- 物理存在：该方法需要证明用户实际拥有该平台。

1.3 futureTPM

1.3.1 可信计算技术面临的挑战

随着移动互联网、量子计算、物联网、云计算、区块链等技术的发展和应用，可信计算技术也面临很多新的应用场景和研究问题[1]：

1) 在移动可信计算方面, 设计包含更小可信计算基 (TCB) 的移动可信体系架构, 以及实现内核运行时和移动应用的安全防护是重要的研究问题;

2) 在可信物联网方面, 轻量级的信任根、高效安全的软件证明、代码安全更新机制是该领域的重要研究问题;

3) 在可信云中, 如何利用虚拟可信平台模块、虚拟机监控技术、新型的硬件安全技术实现云平台安全防护是云厂商们关注的重要安全问题;

4) 在可信区块链方面, 新型的可信执行环境技术可以为区块链提供新的思路, 例如利用硬件安全机制改进共识协议, 使用可信执行环境 (TEE) 保障区块链的计算环境等;

5) 随着量子计算的发展, 设计高效的抗量子密码算法和协议是一个亟需解决的科学问题; 更进一步, 需要设计具有抗量子能力的可信计算模块, 并且构建抗量子可信计算技术体系。

1.3.2 抗量子可信计算研究与 futureTPM

随着量子计算理论的发展, 部分经典模型下的计算困难问题可以在量子计算模型下得到有效或高效求解, 现有基于传统数学困难问题的密码算法和协议将面临严峻的挑战。1994 年 Shor 提出了可以在多项式时间内有效求解大整数分解问题和离散对数问题的量子计算模型, 使得工业界广泛使用的公钥密码算法与密钥交换协议从理论上变得不再安全。2019 年谷歌声称, 在具有 53 个量子比特位的量子计算机上约 200 s 完成的任务在传统计算机上需要执行一万年, 显示出量子计算机相比于传统图灵计算机根本性的优势。

目前国际上对抗量子可信计算平台模块的研究，以 FutureTPM 项目最具代表性。FutureTPM 是欧盟“地平线 2020 计划”资助的一项前沿性研究项目，旨在设计和开发具有抗量子能力（QR）的可信计算平台模块（TPM）。其主要目标是实现从现有广泛应用的 TPM 系统到未来具有抗量子功能的系统的平稳过渡。（欧盟“地平线 2020 计划”- Horizon 2020“欧盟科研框架计划”始于 1984 年，以研究国际前沿和竞争性科技难点为主要内容，是欧盟成员国共同参与的中期重大科研计划）。

抗量子可信计算技术体系在架构上与传统可信计算技术体系类似，不同之处在于抗量子可信计算技术体系需要以抗量子密码算法和协议实现相应的可信计算平台功能。此外，还需要考虑到传统体系的兼容性以及更高需求的计算性能。futureTPM 的愿景是结合强大且经过正式验证的 QR 加密原语提供新一代基于 TPM 的解决方案。其目标是实现平稳过渡来自当前 TPM 环境，基于广泛使用和标准化的密码技术，到系统地通过 QR 密码功能提供增强的安全性，包括安全身份验证、加密和签名功能。futureTPM 将通过为诸如密钥协议之类的原语设计高安全性 QR 算法的创新组合，包括加密、签名、哈希、消息验证码（MAC）功能和直接匿名认证（DAA）等，使 FutureTPM 系统能够生成更安全的信任根，并可适用于与云服务交互、访问公司服务、执行银行业务和电子商务交易，以及广泛的其他服务。

1.3.3 futureTPM 工作组与主要目标

- **为 TPM 设计的安全量子抗密码算法**：FutureTPM 旨在为 TPM 支持的每个加密原语设计和开发相应的可安全证明的 QR 算法，包括对称密码学，非对称密码学密码学和隐私保护原语，如直接匿名认证。

- **使用形式安全分析的设计验证**：FutureTPM 旨在定义和设计形式化方法，包括计算机辅助证明系统和自动证明工具，以支持在 TPM 规范的范围内对系统进行推理所需的安全分析模型。 硬件、软件和虚拟 TPM 的实现** FutureTPM 旨在证明已识别的 QR 算法在各种可能的 TPM 环境中的适用性。这需要在三种类型的 TPM 环境中实现和严格评估所设计的 QR 算法套件：硬件 TPM（hTPM），软件 TPM（sTPM），以及虚拟 TPM（vTPM）。

- **TCG、ISO/IEC 和 ETSI 内的标准化**：该项目的计划成果包括制定标准化提案，推动密码学和 TPM 本身领域的最新技术在涉及相关标准机构的技术委员会中的演进，特别是 ISO、IEC、ETSI 和 TCG。

- **提供运行时风险评估和脆弱性分析方法**：在许多情况下，安装有 TPM 的设备（例如主板）上的操作可能会泄露可使用的敏感信息（例如，通过侧信道攻击），进而成功发起攻击以恢复机密信息。在这种情况下，Future TPM 将设计针对所有系统开发生命周期的各个阶段，包括从设计时到新识别的攻击的实时风险的量化评估和分析。

2. 可信计算标准化

标准是技术产业化应用成熟的标志、也是技术广泛应用的保障。经过 20 多年的探索发展，可信计算技术已形成成熟的标准体系。

- 国际方面，可信计算标准主要由国际可信计算组织（TCG，Trusted Computing Group)维护与发布。
- 国内方面，可信计算标准主要由参与可信计算的产学研机构撰写和维护，形成了覆盖可信根、可信平台、主板、可信软件栈、网络、测评及应用的标准化体系。

本节主要介绍以 TCG 为代表的国际可信计算标准体系和国内可信计算标准体系。

2.1 国内可信计算标准

当前国内可信计算标准已构建了密码为基础、芯片为支柱、主板为平台、软件为核心、网络为纽带、应用成体系的可信计算标准化体系，为落实《网络安全法》，支持《网络安全等级保护条例》、《关键信息基础设施条例》执法，促进“网络安全等级保护”系列标准的实施提供了重要依据和支撑。

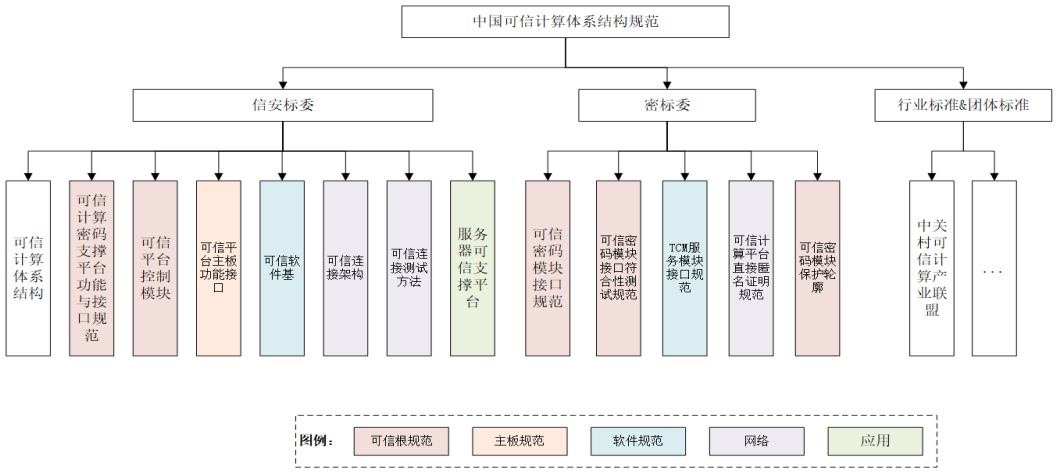


图 2-1-1 国内可信计算标准化体系

2.1.1 国家标准化委员会发布

全国信息安全标准化技术委员会

表 2-1-1 信安标委发布的可信计算标准

标准编号	标准名称	归口单位	标准概述
GB/T 38638-2020	信息安全技术 可信计算 可信计算体系结构	全国信息安全标准化技术委员会	该标准规定了可信计算的体系结构、可信部件及完整性度量模式、可信计算节点类型。 该标准为首次发布。

标准编号	标准名称	归口单位	标准概述
GB/T 29829-2022	信息安全技术 可信计算密码支撑平台功能与接口规范	全国信息安全标准化技术委员会	该标准给出了可信计算密码支撑平台的体系框架和功能原理，规定了可信密码模块的接口规范。该标准替代 GB/T 29829-2013
GB/T 40650-2021	信息安全技术 可信计算规范 可信平台控制模块	全国信息安全标准化技术委员会	该标准描述了可信平台控制模块在可信计算平台中的位置和作用，规定了可信平台控制模块的功能组成、功能接口、安全防护、运行维护要求和证实方法。该标准为首次发布。
GB/T 29827-2013	信息安全技术 可信计算规范 可信平台主板功能接口	全国信息安全标准化技术委员会	该标准规定了可信平台主板的组成结构、信任链构建流程和功能接口。
GB/T 37935-2019	信息安全技术 可信计算规范 可信软件基	全国信息安全标准化技术委员会	该标准规定了可信软件基的功能接口、工作流程、保障要求和交互接口规范。该标准为首次发布。
GB/T 29828-2013	信息安全技术 可信计算规范	全国信息安全标准化技术委员会	该标准规定了可信连接架构的层次、实体、组件、接口、实现流程、评估、隔离和修补以及各个接口的具体实现，解决终端连接到网络的双向用户身份鉴别和平台

标准编号	标准名称	归口单位	标准概述
	可信连接架构		鉴别问题，实现终端连接到网络的可信网络连接。 本标准适用于具有可信平台控制模块的终端与网络的可信网络连接。
GB/T 38644-2020	信息安全技术 可信计算 可信连接测试方法	全国信息安全标准化技术委员会	该标准依据 GB/T 29828-2013，规定了可信网络连接协议以及所涉及的密码算法的测试要求及方法。
GB/T 36639-2018	信息安全技术 可信计算规范 服务器可信支撑平台	全国信息安全标准化技术委员会	该标准规定了服务器可信支撑平台的功能和安全要求，并描述了服务器可信支撑平台的组成结构。 该标准为首次发布。

国家标准化管理委员会

表 2-1-2 国家标准化管理委员会发布的可信计算标准

标准编号	标准名称	归口单位	标准概述
GB/T 30847.1-2014	系统与软件工程 可信计算平台可信性度量 第 1 部分：概述与词汇	国家标准化管理委员会	对可信计算平台可信性度量进行概述；总述可信计算平台及可信计算平台可信性度量方法；简要描述 Plan-Do-Check-Act(PDCA)循环过程;定义可信计算平台可信性度量标准中使用的词汇。
GB/T 30847.2-2014	系统与软件工程 可信计算平台可信性度量	国家标准化管理委员会	从信任链的角度定义了可信计算平台的可信度量方法，包括信任链度量模型和度量指标。

标准编号	标准名称	归口单位	标准概述
	第 2 部分：信任链		

2.1.2 国家密码管理局标准化委员会发布

我国可信计算密码应用相关技术标准于 2007 年开始起步，目前已经形成比较完善的可信密码模块（Trusted Cryptography Module，简称 TCM）标准体系，TCM 以 SM2、SM3、SM4 商用密码算法为核心，相关行业应用标准包括：

表 2-1-3 密标委发布的可信计算标准

标准编号	标准名称	归口单位	标准概述
GM/T 0011-2012	可信计算 可信密码支撑平台功能与接口规范	密码行业标准化技术委员会	描述可信计算密码支撑平台功能原理与要求，并详细定义了可信计算密码支撑平台的密码算法、密钥管理、证书管理、密码协议、密码服务等应用接口规范。该标准在 2012 年发布，2013 年升级成国标（GB/T 29829-2013）。目前该标准在修订中，修订后的标准与 GM/T 0012-2020 兼容。
GM/T 0012-2020	可信计算 可信密码模块接口规范	密码行业标准化技术委员会	该标准描述了可信密码模块的功能，详细定义了可信密码模块的命令接口。该标准替代 GM/T 0012-2012。
GM/T 0013-2021	可信计算 可信密码模块接口	密码行业标准化技术委员会	该标准以《可信计算 可信密码支撑平台接口规范》（GM/T 0011-2012）为基础，规范了可信密码模块的接口命令测试向量、测试方法与测试脚本。该标准在修

	符合性测试规范		订中，修订后会兼容最新的 GM/T 0011。
GM/T 0058-2018	可信计算 TCM 服务模块接口规范	密码行业标准化技术委员会	该标准规定了 TCM 服务模块的组成和接口标准，包含 TSP、TCS 和 TDDL，是面向 TCM 应用层的接口标准。 该标准目前正在修订中，修订后兼容 GM/T 0012-2020，修订牵头单位为 SIG 成员国民技术股份有限公司。
GM/T 0079-2020	可信计算平台直接匿名证明规范	密码行业标准化技术委员会	该标准规定了可信计算平台的直接匿名证明协议的功能、接口和数据结构。 该标准为首次发布。
GM/T 0082-2020	可信密码模块保护轮廓	密码行业标准化技术委员会	该标准以 GB/T 29829 和 GB/T 18336 为基础，构建可信密码模块的保护轮廓，对符合评估保障级第 3 级的 TOE 的定义、安全环境、安全目的、安全要求等进行了详细的说明，并给出相应的基本原理说明。

目前可信计算标准已由 TCM 1.0 全面向 TCM 2.0 演进，密标委组织制定了支撑可信计算密码应用的机制、协议，接口的相关标准。我国 SM2 密码算法的密钥生成、SM2 密码算法签名/验签，SM2 密码算法密钥交换，SM3 密码算法，SM4 密码算法应用等已经被纳入 ISO/IEC 11889 国际可信计算标准。

2.1.3 国内行业与团体发布

表 2-1-4 国内行业与团体发布的可信计算标准（部分）

标准编号	标准名称	归口单位	标准概述
GA/T 2001-2022	移动警务 可信计算总体技术要求	公安部计算机与信息处理标准化技术委员会	该标准规定了移动警务可信计算总体架构、技术要求和密码使用要求。
T/ZTCIA 001-2022	可信计算产品规范	中关村可信计算产业联盟 标准化委员会	N/A
T/ZTCIA 002-2022	网络安全产品安全可信要求》	中关村可信计算产业联盟 标准化委员会	N/A
T/ZTCIA 003-2022	嵌入式可信计算技术要求与测评方法》	中关村可信计算产业联盟 标准化委员会	N/A
T/ZTCIA 004-2022	外设产品可信计算技术规范》	中关村可信计算产业联盟 标准化委员会	N/A

2.1.4 可信计算密码产品认证

在由市场监管总局、国家密码管理局联合发布的《商用密码产品认证目录》第一批/第二批中，包括如下可信计算密码相关产品种类：

表 2-1-5 商用密码产品认证中可信计算产品相关的认证

产品种类	产品描述	认证依据
可信计算密码支撑平台	采取密码技术，为可信计算平台自身的完	● GM/T 0011《可信计算密码支撑平台功能与接口规范》；

产品种类	产品描述	认证依据
	整性、身份可信性和数据安全性提供密码支持。其产品形态主要表现为可信密码模块和可信密码服务模块。	● GM/T 0012《可信计算可信密码模块接口规范》； ● GM/T 0058《可信计算 TCM 服务模块接口规范》； ● GM/T 0028《密码模块安全技术要求》。
可信密码模块	可信计算密码支撑平台的硬件模块，为可信计算平台提供密码运算功能，具有受保护的存储空间。	● GM/T 0012《可信计算可信密码模块接口规范》； ● GM/T 0028《密码模块安全技术要求》。
安全芯片	含密码算法、安全功能，可实现密钥管理机制的集 成电路芯片。	GM/T 0008《安全芯片密码检测准则》。

- **可信密码模块认证**：为贯彻落实《中华人民共和国密码法》，进一步健全完善商用密码产品认证体系，更好满足商用密码产业发展需要，市场监管总局、国家密码管理局于 2022 年 7 月 10 日发布了商用密码产品认证目录第二批，可信密码模块在该目录中。关于可信密码模块的认证，依据 GM/T 0012《可信计算可信密码模块接口规范》和 GM/T 0028《密码模块安全技术要求》进行认证。关于可信密码模块功能接口，按照 GM/T 0012《可信计算可信密码模块接口规范》进行测试；关于密码模块的安全性，按照 GM/T 0028《密码模块安全技术要求》进行认证，可以根据实际需要

选择安全等级 1～4 级进行认证。当前最新的可信计算可信密码模块接口规范为 GM/T 0012-2020 《可信计算可信密码模块接口规范》,也就是 TCM2.0 标准，其协议为 TPM2.0 协议的一个子集，在算法上只支持中国算法 SM2/SM3/SM4。

- **可信计算密码支撑平台认证：**在《商用密码产品认证目录（第一批）》中，有《可信计算密码支撑平台》产品认证类别，依据 GM/T 0011/0012/0058/0028 进行认证。GM/T 0028 分为 1-4 安全等级进行认证。
- **安全芯片认证：**单颗芯片实现的 TCM 可信密码模块属于密码安全芯片类，在《商用密码产品认证目录（第一批）》中，安全芯片依据《GM/T 0008-2012 安全芯片密码检测准则》来检测，分为 1-4 安全等级进行认证。

2.1.5 可信计算产品认证

由中关村可信计算产业联盟和公安部第三研究所联合开展的可信计算产品联合认证，认证范围包括如下产品种类：

表 2-1-6 中关村可信计算产业联盟开展的可信计算产品认证

产品种类	认证依据
可信服务器	T/ZTCIA 001-2022 可信计算产品规范
可信网络安全产品	T/ZTCIA 002-2022 可信计算 网络安全产品安全可信要求
办公设备产品	T/ZTCIA 003-2022 可信计算 办公设备产品技术规范

2.2 国际可信计算标准

2.2.1 ISO 发布

ISO 简介

ISO（国际标准化组织）和 IEC（国际电工委员会）组成了专门的全球标准化体系。作为 ISO 或 IEC 成员的国家机构通过各自组织建立的技术委员会参与国际标准的制定，以处理特定领域的技术活动。ISO 和 IEC 技术委员会在共同感兴趣的领域进行合作。与 ISO 和 IEC 保持联系的其他政府和非政府国际组织也参加了这项工作。在信息技术领域，ISO 和 IEC 成立了一个联合技术委员会 ISO/IEC JTC 1。国际标准是根据 ISO/IEC 指令第 2 部分中给出的规则起草的。联合技术委员会的主要任务是制定国际标准。联合技术委员会通过的国际标准草案分发给国家机构表决。作为国际标准出版需要至少 75%的国家机构投票通过。

ISO/IEC 11889 系列标准概述

TPM2.0 库规范系列标准以 ISO/IEC 11889:2015 发布。该规范由可信计算组（TCG）提交给 ISO/IEC JTC 1*，遵循 JTC 1 公开可用规范(PAS)转换过程。在最终的 TPM 2.0 标准化投票中，来自发达经济体和新兴经济体的支持，赞成票来自澳大利亚、比利时、加拿大、中国、捷克、丹麦、芬兰、法国、加纳、爱尔兰、意大利、日本、韩国、黎巴嫩、马来西亚、荷兰、尼日利亚、挪威、俄罗斯联邦、南非、阿拉伯联合酋长国、英国和美国。 TPM 2.0 库规范支持现代安全和隐私保护，该规范整合了基于硬件和基于软件的安全技术的优势。它在计算设备中的实现保护了加密密钥；防止私钥被导出；屏蔽用于身份验证的 PIN 值；并记录和匿名报告在启动过

程中加载的软件，以防止恶意软件和攻击。因此，它将成为任何综合安全战略的重要组成部分。

ISO/IEC 11889 系列标准

- ISO/IEC 11889-1:2015 Information technology – TPM Library – Part 1: Architecture
- ISO/IEC 11889-2:2015 Information technology – TPM Library – Part 2: Structures
- ISO/IEC 11889-3:2015 Information technology – TPM Library – Part 3: Commands
- ISO/IEC 11889-4:2015 Information technology – TPM Library – Part 4: Supporting Routines

2.2.2 TCG 发布

关于 TCG

可信计算组（Trusted Computing Group，TCG）是一个非营利组织，旨在为可互操作的可信计算平台开发、定义和推广基于硬件信任根的供应商无关的全球行业规范和标准。TCG 的核心技术包括可信平台模块（Trusted Platform Module，TPM），可信网络通信（Trusted Network Communications，TNC）以及网络安全和自加密驱动器的规范和标准。TCG 还设有工作组，将信任的核心概念从企业扩展到物联网的云安全、虚拟化和其他平台以及计算服务。

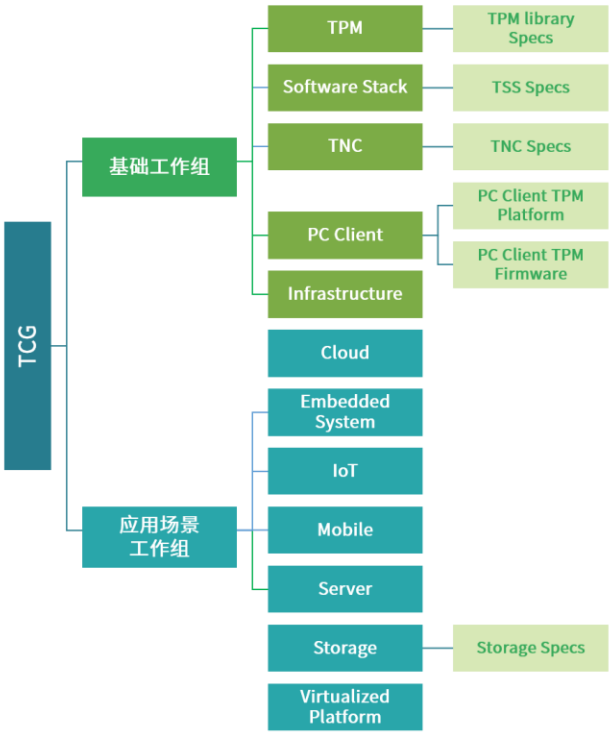


图 2-2-1 TCG 工作组

TPM2.0 库标准

TPM2.0 库标准是 TPM2.0 中基础规范，该规范描述了 TPM2.0 实现的所有核心功能。库标准由四部分组成：

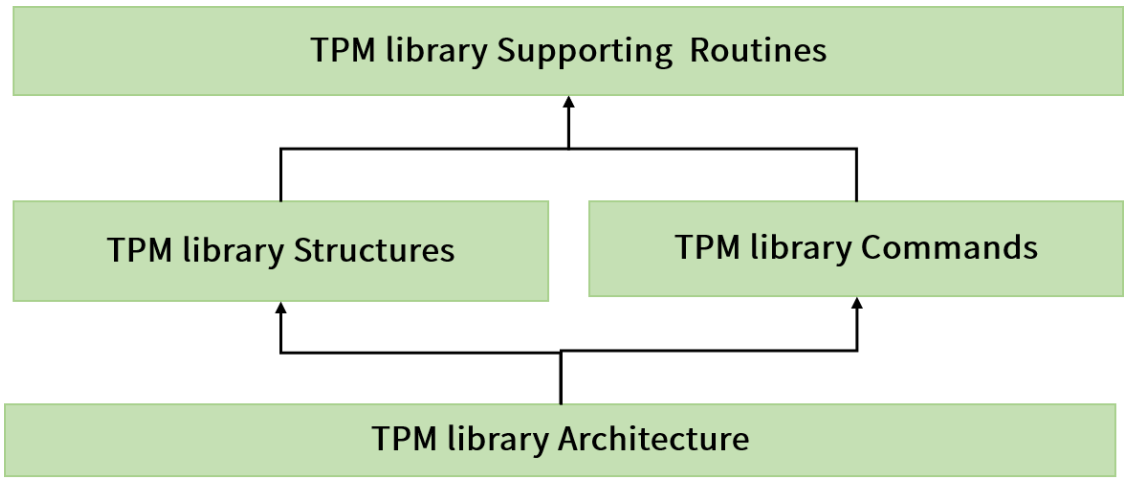


图 2-2-2TPM 库标准

- 第一部分：架构规范，该部分描述了 TPM2.0 的操作规范、设计原理及工作细节（例如如何创建用于授权、审计、加密命令的会话）。
- 第二部分：数据结构规范，该部分描述了 TPM2.0 所使用的数据类型、结构体、联合体，该部分也包含了 TPM 所使用的命令码、错误码等内容。
- 第三部分：命令规范，该部分描述了 TPM2.0 中所使用的命令及其功能使用方法，包括了命令的输入输出参数、参考的错误码含义，并提供了命令在 TPM 中实现的伪代码。
- 第四部分： 支持例程，该部分描述了 TPM2.0 的实现细节（可以作为 tpm 模拟器实现的设计参考）。

TCG TPM 库标准下载官网入口

- Part 1: Architecture： https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf
- Part 2: Structures：https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part2_Structures_pub.pdf
- Part 3:Commands：https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part3_Commands_pub.pdf
- Part 3:Commands - Code：https://trustedcomputinggroup.org/wpcontent/uploads/TCG_TPM2_r1p59_Part3_Commands_code_pub.pdf
- Part 4:Supporting Routines：https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part4_SuppRoutines_pub.pdf

- Part 4:Supporting Routines – Code：https://trustedcomputinggroup.org/wpcontent/uploads/TCG_TPM2_r1p59_Part4_SuppRoutines_code_pub.pdf

TPM Software Stack（TSS）标准

TSS（TCG Software Stack）是一种软件规范，提供了访问 TPM 功能的标准 API，TSS 由多层组成，允许对可扩展的 TSS 实现进行定制，以适应高端系统和资源受限的低端系统。TSS 也设计了应用程序提供与本地或远程 TPM 通信的接口方法。

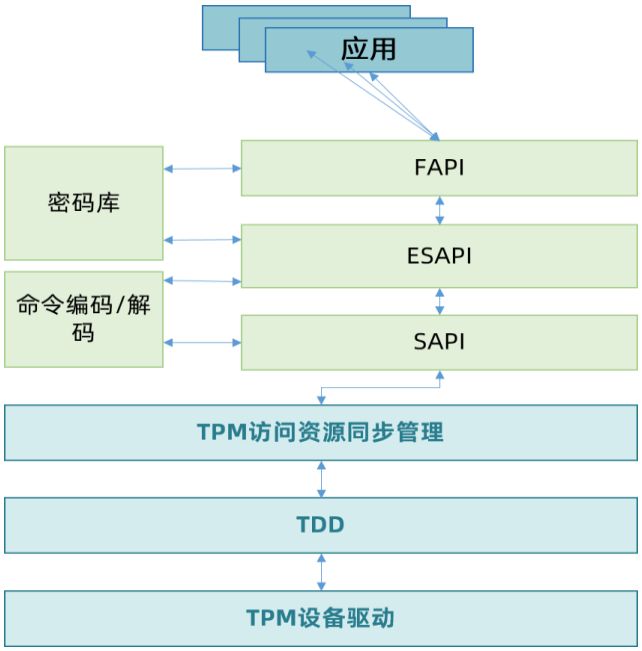


图 2-2-3 TSS2.0 架构图

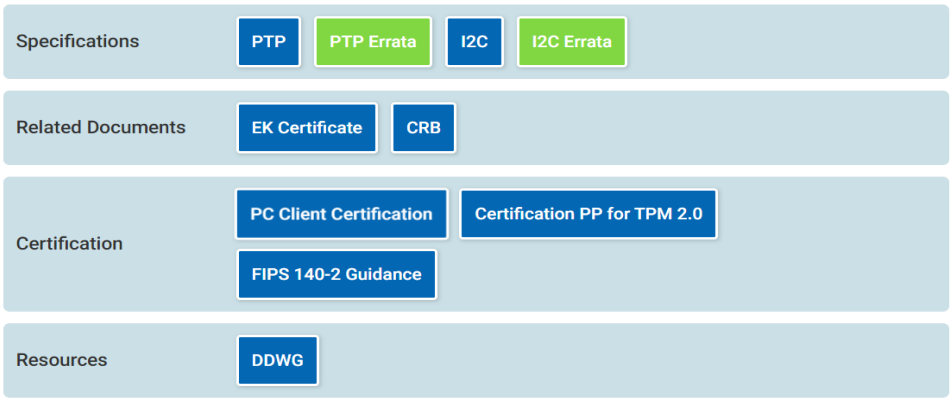
应用程序开发人员可以参考该规范开发可互操作的客户端应用程序，以实现对可信计算能力的调用。TSS 的另一个作用是将程序程序员与与 TPM 接口的底层细节隔离开来，降低 TPM 应用开发的学习成本。

- TSS 规范官网入口：https://trustedcomputinggroup.org/resource/tcg-software-stack-tss-specification/

PC Client

PC Client 系列规范主要定义了 TCG 技术在计算机（例如笔记本电脑、台式机或平板电脑）中的上下文中所呈现的功能和行为，包括 TPM 以及与 TPM 交互或在其平台中集成 TPM 的平台 OEM 和固件供应商所需参考的规范、技术要求和指导。主要分为 PC Client TPM Platform 标准及配套参考文档和 PC Client Firmware 标准及配套参考文档。

PC Client TPM Platform 标准及配套文档体系:



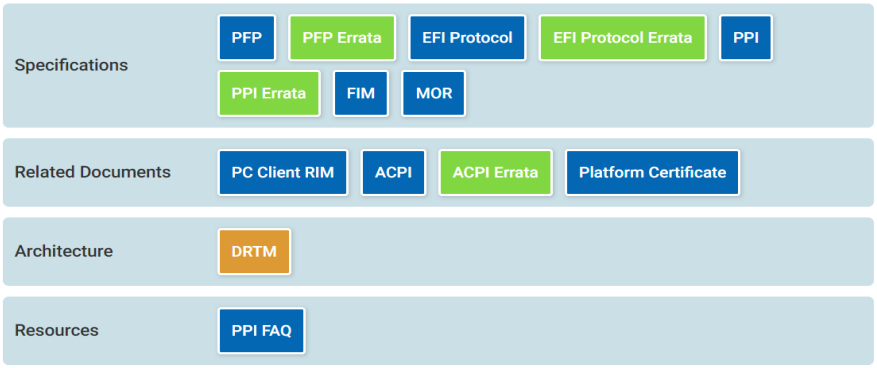
2-2-4 PC Client TPM Platform 标准及配套文档体系

注：附图源自 TCG 官网

缩略语

- PTP – Platform TPM Profile
- CRB – Command Response Buffer interface
- DDWG – Device Driver’ s Writers Guide
- Certification PP – Certification Protection Profile
- TIS – TPM Interface Specification

PC Client Firmware 标准及配套文档体系:



2-2-5 PC Client Firmware 标准及配套文档体系

注：附图源自 TCG 官网

缩略语

- PFP – Platform Firmware Profile
- PPI – Physical Presence Interface
- FIM – Firmware Integrity Measurement
- MOR – Reset Attack Mitigation
- Memory on reset attack mitigation
- RIM – Reference Integrity Manifest
- DRTM – Dynamic Root of Trust for Measurement

表 2-2-1 TCG PC Client 系列标准

类型	标准名称/案例	最新版本及发布时间	标准概述
PC Client TPM Platform	TCG PC Client Platform TPM Profile (PTP)	v1.05 Rrvision14 2020.9.4	A TPM claiming adherence to this specification SHALL be compliant with the TPM Library Specification

类型	标准名称/案例	最新版本及发布时间	标准概述
	Specification for TPM2.0		
PC Client TPM Platform	TCG TPM I2C Interface Specification for TPM2.0	Revision 1.0 2016.10.7	The Trusted Computing Group TPM I2C Interface Specification is an industry specification that defines an I2C Interface for TPM 2.0.
PC Client Firmware	TCG PC Client Specific Platform Firmware Profile Specification	Version 1.05 Revision 23 2021.5.7	The PC Client Platform Specific Profile for TPM 2.0 systems defines the requirements for platform firmware to initialize and interact with a TPM 2.0 device in a PC Client platform
PC Client Firmware	TCG EFI Protocol Specification for TPM2.0	Version1.0 Revision 0.13 2016.3.30	It defines data structures and APIs that allow an OS to interact with UEFI firmware to query information important in an early OS boot stage. Such information include: is a TPM present, which PCR banks are active, change active PCR banks, obtain the TCG boot log, extend hashes to PCRs, and append events to the TCG boot log.
PC Client Firmware	TCG Platform Reset Attack Mitigation Specification	Version 1.1 Revision 17 2019.2.21	When a platform reboots or shuts down, the contents of volatile memory (RAM) are not immediately lost. Without an electric charge to maintain the data in memory, the data will begin to decay. During this period,

类型	标准名称/案例	最新版本及发布时间	标准概述
			there is a short timeframe during which an attacker can turn off or reboot the platform, and quickly turn it back on to boot into a program that dumps the contents of memory. Encryption keys and other secrets can be easily compromised through this method.
PC Client Firmware	TCG PC Client Platform Firmware Integrity Measurement	Version 1.0 Revision 43 2021.5.7	This document describes the requirements for a PC Client Endpoint in an enterprise computing environment complying with SP 800-155 BIOS Integrity Measurements
PC Client Firmware	TCG PC Client Physical Presence Interface Specification	Version 1.30 Revision 0.52 2015.7.28	This specification defines an interface between an operating system and the firmware to manage the configuration of a TPM and, if required, initiate TPM related operations. The specification gives suggestions on UI wording for interactions with users of a system, if UI interaction is required.

Storage

存储工作组以现有的 TCG 技术和理念为基础，重点关注专用存储系统上的安全服务标准。其中一个目标是开发标准和实践，用于跨专用存储控制器接口定义相同

的安全服务，包括但不限于 ATA、串行 ATA、SCSI、FibreChannel、USB 存储、IEEE 1394、网络附加存储（TCP/IP）、NVM Express 和 iSCSI。存储系统包括磁盘驱动器、可移动媒体驱动器、闪存和多个存储设备系统。 存储标准体系如下图所示。

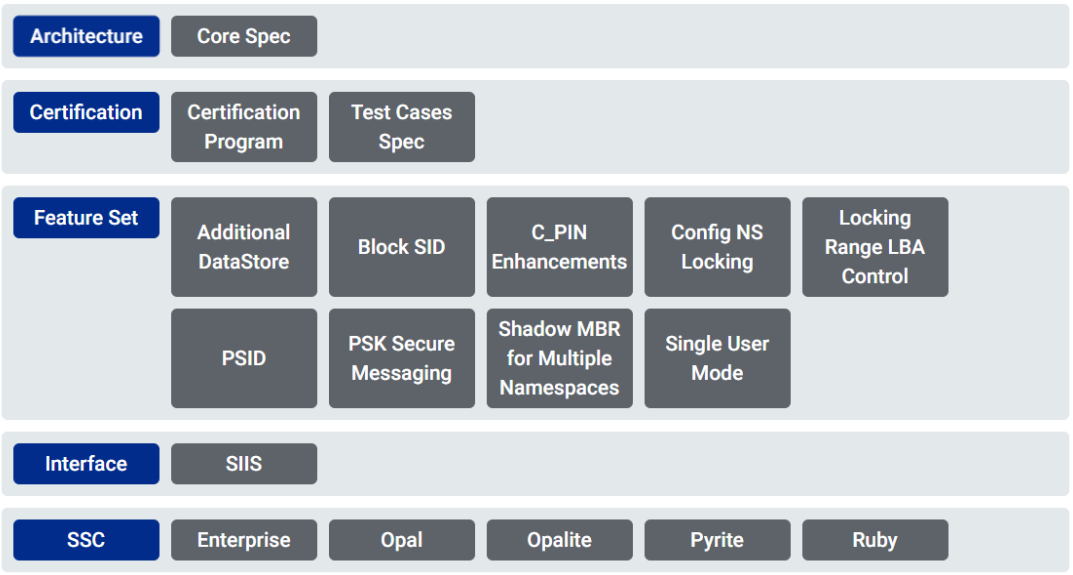


图 2-2-6 TCG 可信存储标准体系

注：附图源自 TCG 官网

TNC（可信网络通信）

TCG 的可信网络通信（TNC-Trusted Network Communications）工作组定义并发布了一个开放架构和一套不断增长的网络安全标准，在跨各种端点、网络技术和策略的多供应商环境中提供可互操作的端到端信任。TNC 支持在不同的网络和安全系统之间进行端点遵从性评估、智能策略决策、动态安全实施和安全自动化。

- 标准编制

TNC 工作组已经定义并发布了一个开放架构和一套不断增长的标准，用于端点遵从性评估、网络访问控制和安全自动化。TNC 体系结构使网络服务提供商能够在网络连接和跨不同安全和网络设备的协调信息时或之后执行有关端点完整性的策略。

表 2-2-2 TCG TNC 系列标准

一级标准	子标准
TNC Architecture for Interoperability	无
IF-IMC – Integrity Measurement Collector Interface	无
IF-IMV – Integrity Measurement Verifier Interface	无
IF-TNCCS – Trusted Network Connect Client-Server Interface	IF-TNCCS: TLV Binding
IF-TNCCS – Trusted Network Connect Client-Server Interface	IF-TNCCS: Protocol Bindings for SoH
IF-M – Vendor-Specific IMC/IMV Messages Interface	IF-M: TLV Binding
IF-M – Vendor-Specific IMC/IMV Messages Interface	SWID Message and Attributes for IF-M
IF-T – Network Authorization Transport Interface	IF-T: Protocol Bindings for Tunneled EAP Methods
IF-T – Network Authorization Transport Interface	IF-T: Binding to TLS
IF-PEP – Policy Enforcement Point Interface	IF-PEP: Protocol Bindings for RADIUS
IF-MAP – Metadata Access Point Interface	IF-MAP Binding for SOAP
IF-MAP – Metadata Access Point Interface	IF-MAP Metadata for Network Security
IF-MAP – Metadata Access Point Interface	IF-MAP Metadata for ICS Security
IF-MAP – Metadata Access Point Interface	MAP Content Authorization

一级标准	子标准
ECP - Endpoint Compliance Profile	无
CESP - Clientless Endpoint Support Profile	无
Server Discovery and Validation	无
Federated TNC	无
IF-PTS - Platform Trust Services Interface	Attestation PTS Protocol: Binding to IF-M
IF-PTS - Platform Trust Services Interface	Simple Object Schema
IF-PTS - Platform Trust Services Interface	Core Integrity Schema
IF-PTS - Platform Trust Services Interface	Integrity Report Schema
IF-PTS - Platform Trust Services Interface	Reference Manifest (RM) Schema
IF-PTS - Platform Trust Services Interface	Security Qualities Schema
IF-PTS - Platform Trust Services Interface	Verification Result Schema

- 应用场景

TNC 不同场景下的安全需求提供了可互操作的标准，TNC 标准确保跨各种端点、网络技术和策略的多供应商互操作性：

表 2-2-3 TCG TNC 应用场景

需求场景	要求	解决的问题
合规遵从	网络和端点可见性	谁和什么都在我的网络上?
合规遵从	端点遵从性	我网络上的设备安全吗?
合规遵从	端点遵从性	用户/设备行为是否合适?
访问控制	网络实施	阻止未经授权的用户、设备或行为
访问控制	网络实施	对授权用户/设备授予适当的访问权限
安全调度	安全系统协调	共享用户、设备、威胁等实时信息。

- 标准推广

TNC 标准的采用也从供应商和最终用户扩展到其他标准组织。互联网工程任务组（IETF）网络端点评估（NEA）工作组发布了几个基于TNC客户端-服务器协议的rfc。

表 2-2-4 IETF 发布的 TNC 应用标准

IETF RFC	TNC Specification
PA-TNC: A Posture Attribute (PA) Protocol Compatible with Trusted Network Connect (TNC) - RFC 5792	TNC IF-M: TLV Binding Version 1.0
PB-TNC: A Posture Broker (PB) Protocol Compatible with Trusted Network Connect (TNC) - RFC 5793	TNC IF-TNCCS: TLV Binding Version 2.0
A Posture Transport Protocol over TLS (PT-TLS) - RFC 6876	TNC IF-T Binding to TLS Version 2.0
PT-EAP: Posture Transport (PT) Protocol for Extensible Authentication Protocol (EAP) Tunnel Methods - RFC 7171	TNC IF-T: Protocol Bindings for Tunneled EAP Methods, Version 2.0

TNC 提供了一个灵活、开放的体系结构，可以适应不断变化的环境，而不依赖于任何一家供应商。跨国公司支持的技术提高了投资回报率，支持使用现有的网络设备和同类最佳产品，并避免了供应商锁定。可见性和协调性有助于有效的网络管理和安全。TNC 开放的网络安全架构和完整的标准得益于安全专家的全面技术审查。为了获得最强的安全性，TNC 可以利用 TPM 进行健壮的身份验证、认证和危害检测。商业供应商、开源社区和 IETF 对 TNC 标准提供了广泛的支持。TNC 可以与 TPM 集成，以实现安全身份验证和认证，解决 rootkit 和其他受损软件的检测和缓解问题。TNC 标准为保护嵌入式系统(如网络设备、汽车和物联网解决方案)提供了通信基础。

其他

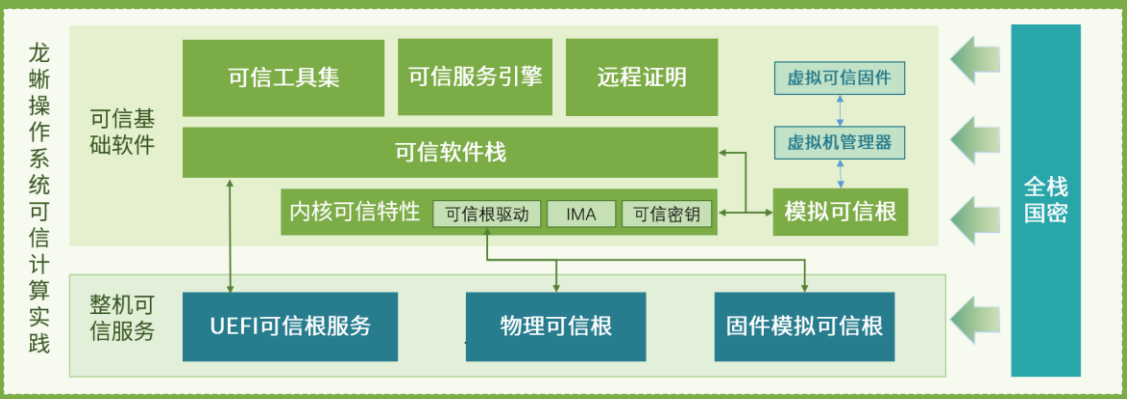
表 2-2-5 TCG 其他工作组发布的标准

工作组	标准名称/案例
cloud	Trusted Multi-Tenant Work Group Trust Assessment Framework
cloud	TCG Trusted Multi-Tenant Infrastructure Use Cases
cloud	Trusted Multi-Tenant Infrastructure Reference Framework
cloud	Cloud Computing and Security - A Natural Match
Infrastructure	TPM 2.0 Keys for Device Identity and Attestation
Infrastructure	TCG Component Class Registry
Infrastructure	TCG EK Credential Profile for TPM Family 2.0
Infrastructure	Canonical Event Log Format
Infrastructure	PCIe-based Component Class RegistryZ
Server	TCG Server Management Domain Firmware Profile Specification
virtualized Platform	Virtualized Trusted Platform Architecture Specification
Cyber Resilient Technologies	
DICE	DICE Endorsement Architecture for Devices
DICE	DICE Attestation Architecture
DICE	DICE Layering Architecture
DICE	Symmetric Identity Based Device Attestation
DICE	DICE Certificate Profiles
Industrial	Industrial Internet Security Framework
Industrial	Standards for Securing Industrial Equipment

工作组	标准名称/案例
Industrial	Industrial Internet Security Framework
Industrial	TNC IF-MAP Metadata for ICS Security
Industrial	TCG Guidance for Securing Industrial Control Systems Using TCG Technology
IoT	TCG Guidance for Secure Update of Software and Firmware on Embedded Systems
IoT	Architect's Guide IoT Security
Measurement AND Attestation Roots	TCG MARS API Specification Version 1, Revision 2
Measurement AND Attestation Roots	MARS Library Specification
Measurement AND Attestation Roots	MARS Use Cases and Considerations
Mobile	TCG Runtime Integrity Preservation in Mobile Devices
Mobile	TCG Trusted Network Communications for Mobile Platforms
Mobile	TCG TPM 2.0 Mobile Specification Implementation Guidance
Embedded Systems	

3. 龙蜥操作系统可信计算实践指南

龙蜥操作系统以开源社区优秀成果为上游集成了模拟可信根、可信软件栈、可信工具集，满足用户对可信计算基础服务的需求。同时，可信计算 SIG 对可信根增强引擎等可信计算基础软件进行了探索分析，形成了龙蜥操作系统可信计算实践指南相关内容。未来，针对可信计算基础软件全栈国密支持与实践探索也将是可信计算 SIG 的重点工作。



3.1 模拟可信根

3.1.1 swtpm 概述

可信平台模块（TPM）是一个提供多种安全功能的组件，例如加密、随机数生成、度量等，由于操作系统安全认证的要求，现在已广泛部署在服务器、PC、终端设备中。对于想要使用 TPM 来开发安全功能的开发人员来说，通过访问主机上的 TPM 硬件，往往会受到 TPM 物理设备访问数量的限制。而软件 TPM 模拟器通常是一个不错的选择，通过使用软件实现的 TPM 模拟器，用户可以轻松地在 TPM 1.2 和 TPM 2.0 之间切换，这使开发人员的使用上更加便捷。

swtpm 项目的目标是提供一个 TPM（TPM1.2 和 TPM2）模拟器，该模拟器可以集成到虚拟化的环境中，如虚拟机和容器。到目前为止，swtpm 已经集成到 QEMU 中，并作为原型集成到 RunC（PR）中。swtpm 的构建基于 libtpms 项目。

3.1.2 龙蜥社区在 swtpm 上的工作与探索

龙蜥社区自其可信计算 SIG 成立以来，一直在关注可信计算业界进展和国际 OSV 厂商的可信计算方案。在完成 swtpm 的适配和实践后，龙蜥社区也将自己的 swtpm 的开发与使用经验写入到白皮书中。未来，龙蜥社区除了继续加强与 swtpm 项目的交流与贡献外，还将结合自己在国密/自主创新/云计算的积累围绕 swtpm 开展一些国密、机密计算相关的工作，敬请期待。

3.1.3 龙蜥 Anolis OS 上 swtpm 实践

Anolis OS 上 swtpm 安装与配置、运行

Anolis OS 中可使用 swtpm 和 libtpm 源码安装或使用 Anolis OS 提供的 yum 源进行安装，安装方式如下。

注：使用源码安装的 swtpm 版本较新，可能与 yum 安装的版本存在冲突，在使用任一安装方法前，请确保系统内不存在冲突版本，以免造成不必要的使用异常。

源码安装

1）swtpm 代码仓库

- swtpm
- libtpms

2）swtpm 编译

```
# 安装依赖包
yum  install -y automake autoconf libtool gcc gcc-c++ make \
openssl-devel pkg-config socat net-tools-deprecated \
libtasn1-devel gnutls gnutls-devel libseccomp-devel \
json-glib-devel expect softhsm
# 下载 libtpms 源码
git clone https://github.com/stefanberger/libtpms
cd libtpms
# 编译并安装 libtpms
./autogen.sh --prefix=/usr --libdir=/usr/lib64 --with-openssl \
--with-tpm2

make -j4
make -j4 check
sudo make install
# 下载 swtpm 源码
git clone https://github.com/stefanberger/swtpm
cd swtpm
# 编译并安装 swtpm
 ./autogen.sh --prefix=/usr --libdir=/usr/lib64 --with-openssl \
--with-tss-user=root --with-tss-group=tss --with-cuse
 make -j4
 sudo make check -j4
 sudo make install
```

Anolis OS yum 安装 swtpm

```
yum install libtpms swtpm swtpm-devel swtpm-tools
```

配置

swtpm with local

swtpm 提供了 3 种类型的接口：socket、chardev 和 cuse。通过 cuse 接口，swtpm 可以在用户空间创建字符设备，用来接收 tpm 指令。

目前 anolis 8.8 yum 源默认安装的 swtpm 不支持 cuse 接口类型，需要通过下载源码，安装方式可参考 swtpm 编译。

swtpm_cuse 需要内核 cuse 内核模块，处理方法如下：

```
# 安装内核 cuse 模块
yum install kernel-modules-extra
modprobe cuse
```

tpm 设备运行

```
#1. 初始 tpm2 state
mkdir /tmp/myvtpm0;
chown -R tss:root /tmp/myvtpm0
swtpm_setup --tpm2 --tpm-state /tmp/myvtpm0

#2. 创建 tpm2 字符设备
export TPM_PATH=/tmp/myvtpm0
swtpm_cuse --tpm2 -n tpm0
#3. 启动 tpm 设备
swtpm_ioctl -i --tpm-device /dev/tpm0
```

确认启动成功

```
[root@localhost swtpm]# tpm2_pcrread
sha1:
sha256:
  0 : 0x0000000000000000000000000000000000000000000000000000000000000000
  1 : 0x0000000000000000000000000000000000000000000000000000000000000000
  2 : 0x0000000000000000000000000000000000000000000000000000000000000000
  3 : 0x0000000000000000000000000000000000000000000000000000000000000000
  4 : 0x0000000000000000000000000000000000000000000000000000000000000000
  5 : 0x0000000000000000000000000000000000000000000000000000000000000000
  6 : 0x0000000000000000000000000000000000000000000000000000000000000000
  7 : 0x0000000000000000000000000000000000000000000000000000000000000000
  8 : 0x0000000000000000000000000000000000000000000000000000000000000000
  9 : 0x0000000000000000000000000000000000000000000000000000000000000000
  10: 0x0000000000000000000000000000000000000000000000000000000000000000
```

```
  11: 0x0000000000000000000000000000000000000000000000000000000000000000
  12: 0x0000000000000000000000000000000000000000000000000000000000000000
  13: 0x0000000000000000000000000000000000000000000000000000000000000000
  14: 0x0000000000000000000000000000000000000000000000000000000000000000
  15: 0x0000000000000000000000000000000000000000000000000000000000000000
  16: 0x0000000000000000000000000000000000000000000000000000000000000000
  17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  23: 0x0000000000000000000000000000000000000000000000000000000000000000
sha384:
sha512:
```

swtpm with QEMU

这里我们选择 swtpm 提供的“socket”类型的接口，因为它是唯一不需要在/dev 中创建节点的类型。首先，在 VM 目录中创建一个目录来存储 TPM 状态：

```
$ mkdir ${path_to_vm}/mytpm0
```

为 qemu 创建 socket file 和 swtpm-socket

```
$ swtpm socket --tpmstate dir=${path_to_vm}/mytpm0 \
        --ctrl type=unixio,path=${path_to_vm}/mytpm0/swtpm-sock \
        --log level=20
```

默认情况下，swtpm 启动 TPM 1.2 模拟器，并将状态存储在目录中的“TPM-00.permall”中。还可以创建 TPM 2.0 实例

```
swtpm socket --tpm2 --tpmstate dir=${path_to_vm}/mytpm0 \
        --ctrl type=unixio,path=${path_to_vm}/mytpm0/swtpm-sock \
        --log level=20
```

为 qemu 添加 swtpm 参数：

- X86_64

```
 -chardev socket,id=chrtpm,path=${path_to_vm}/mytpm0/swtpm-sock \
-tpmdev emulator,id=tpm0,chardev=chrtpm \
-device tpm-tis,tpmdev=tpm0
```

- aarch64

```
-chardev socket,id=chrtpm,path=${path_to_vm}/mytpm0/swtpm-sock \
-tpmdev emulator,id=tpm0,chardev=chrtpm \
-device tpm-tis-device,tpmdev=tpm0
```

swtpm with Libvirt

如果使用 libvirt， 在虚拟机的 xml 文件中插入如下内容

```
<devices>
    <tpm model='tpm-tis'>
      <backend type='emulator' version='2.0'/>
    </tpm>
 </devices>
```

➢ 启动虚拟机

启动虚拟机前需要使用chmod命令给目录/var/lib/swtpm-localca/赋予如下权限，否则 libvirt 无法拉起 swtpm。

```
chmod -R 777 /var/lib/swtpm-localca/
virsh start vm
```

确认度量启动使能成功

度量启动功能使能与否由虚拟机 BIOS 决定，目前龙蜥 OS Anolis OS8.8 版本中的虚拟机 uefi 固件已经具备了度量启动的能力。若宿主机采用其他版本的 edk2 组件，请确认其是否支持度量启动功能。

使用 root 用户登录虚拟机，确认虚拟机中是否安装了 tpm 驱动、tpm2-tss 协议栈及 tpm2-tools 工具。 龙蜥 OS Anolis OS8.8 版本中默认安装了 tpm 驱动(tpm_tis.ko)、tpm2-tss 协议栈和 tpm2-tools 工具。若使用其他操作系统，可以使用如下命令检查是否安装了驱动和相关工具

```
# lsmod |grep tpm
# tpm_tis          16384   0
```

```
#
# yum list installed | grep -E 'tpm2-tss|tpm2-tools'
#
# yum install tpm2-tss tpm2-tools
```

可以使用 tpm2_pcrread 命令列出所有的 pcr 值。

```
[root@localhost ~]# tpm2_pcrread
sha1:
  0 : 0xB88919A8FA33C7A11CEB80A1B9772B499BDAABC8
  1 : 0xED92EDC2A5E26D77F83020956E1AA02140870AC3
  2 : 0xB2A83B0EBF2F8374299A5B2BDFC31EA955AD7236
  3 : 0xB2A83B0EBF2F8374299A5B2BDFC31EA955AD7236
  4 : 0x30DDAE4ED835392D81A7CE6FEF905E169BAC27A5
  5 : 0x7BC897262CAD4E3F16F5CE180F6F4B6DEE253483
  6 : 0xB2A83B0EBF2F8374299A5B2BDFC31EA955AD7236
  7 : 0xC8D7DB36A45078BA06A86DAD3A20DCFD525C1E1B
  8 : 0x26B98CA9A67B20C4E7B9C1DAFC6890234CBF6E38
  9 : 0x92B9BA924DEBF7A64AB157689AE4AC921B9E930D
  10: 0x96E96D79512639B9A2DF577CE237D18F544BD74D
  11: 0x0000000000000000000000000000000000000000
  12: 0x0000000000000000000000000000000000000000
  13: 0x0000000000000000000000000000000000000000
  14: 0x8DF12380EDE005407EAB81DA4405321E0DA61280
  15: 0x0000000000000000000000000000000000000000
  16: 0x0000000000000000000000000000000000000000
  17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  23: 0x0000000000000000000000000000000000000000
sha256:
  0 : 0xE3B7A76FDC83187F0233F4616FED23301B044DE62AABC0CCADE6D9468FCB4233
  1 : 0x862224F4F2B87A4DF717EB92BB828C4598C4CF411ADF83FC9BB084B6D31A5D09
  2 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
  3 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
  4 : 0x8D870A781FF79622E72E858B36F025428C15846ABB3D54E0EEACA33E418B9E91
  5 : 0x60EB17AA48B50CC8E78C052BAA633B0848F36B452FC4BE6C2481B525E595C8C8
  6 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
  7 : 0x97FB0EDBE28C25A14D775090F36682D04596FDA4BF5750F275F76C1643BBDC2D
  8 : 0x62A10C0A8638B71A355AAB7C8C66BFE052EBF2F1E9C5308A430AF5FC7652B35E
  9 : 0xED743F4D59ABE8C055EA0E8CE983879D69DBE9894F42172ECFAA65E8583E9DFF
  10: 0xF4FA23203592F54BD5E4392C84CD9591D5D8211638D128CCBE332F54BDD287B0
  11: 0x0000000000000000000000000000000000000000000000000000000000000000
  12: 0x0000000000000000000000000000000000000000000000000000000000000000
  13: 0x0000000000000000000000000000000000000000000000000000000000000000
  14: 0xA4DAD77FB3B6CACBD20F556986C5D917F5E322C123AF82D12C5E5B7EF7AE9938
  15: 0x0000000000000000000000000000000000000000000000000000000000000000
  16: 0x0000000000000000000000000000000000000000000000000000000000000000
  17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

```
  22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  23: 0x0000000000000000000000000000000000000000000000000000000000000000
sha384:
  0 : 0xE0B9E19988D06E4774A33A802981E77123045D56492146A914331AA1FA49AA99DE549823515E6D862779E2F959FF5AC6
  1 : 0x1C44C67D8DD3C86ABD4BAFCC6761DDFFDA96B843F271C6D4D92F84AA8C11BF205831F33D57FB4E960A9C0E83D5C32827
  2 : 0x518923B0F955D08DA077C96AABA522B9DECEDE61C599CEA6C41889CFBEA4AE4D50529D96FE4D1AFDAFB65E7F95BF23C4
  3 : 0x518923B0F955D08DA077C96AABA522B9DECEDE61C599CEA6C41889CFBEA4AE4D50529D96FE4D1AFDAFB65E7F95BF23C4
  4 : 0xBF5307DF2DC437D1F9CB35CB1A85E00717F150C306F01ED7D1EE3565E4626242AE41E9F2F1EDAD9C3F85A34F54F1C172
  5 : 0xAAA365D0D07B6C656D4F8A78346951ECFC2D7C92D3EE475925D9900BB22A255BFFB01B3C9E5CE631CD9BB3C91BB868DE
  6 : 0x518923B0F955D08DA077C96AABA522B9DECEDE61C599CEA6C41889CFBEA4AE4D50529D96FE4D1AFDAFB65E7F95BF23C4
  7 : 0x8742CE00FA4AAB6A8C3B30584D1BB01D4BB680CD9D72923DDCD3600B25EDB9BE9B13B4714A023AE7DC57003ACFB544C1
  8 : 0xADC524F78EAE447F5068D5FCBFF0C9E235CA9903D91FCF21A753A5F7E30B50445C67D7B14184C202C56FFB0BCD55EE3A
  9 : 0xC4F3002193E307C45F62DB79640F3EE54F4738EF83C138010FCA9E47BAD92FACEC0DCADFD0A7E9AB17EDA10F772F5A66
  10: 0x97F83AABAE79094226377E6288AD64BE6A7BEE26FB40E1846D7A2A877F569633E65CAF72C02DE0665AFE626F476A6124
  11: 0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  12: 0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  13: 0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  14: 0x108357C39E8DC8F150A33738567AF451908F80DDFC8C14801FBD513F307DA99082EA0ABA8CC7E042940F310E54C8AB10
  15: 0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  16: 0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  23: 0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
sha512:
  0 : 0x0893229AF987740D78145E551F2C04D6C3A1C3FD4043EBB6BCF6C94F8EDF92EE99DC44916C6FF3AB4F4492EA3AA6C1D60C9912DD379E3B9CF9E7BCF08789720C
  1 : 0x2CCA44A4B710CFFA20EF4C10F378E63E4D2065462A69981381C9DB1BC1D0D8396A09480CC31B4102E6C29A1F3002E170B52E1BE2FEE80E565146C04A534A4BDD
  2 : 0x27EC091533C4B9EEA38DD14C3A3ECDEF0A99C1E564CBE66DFE008250154E7839B0B75228FE8DEBCC4CA330E6AEBC1ABC74070BC9C9C1E26B939C9D916E45E13C
```

```
  3 : 0x27EC091533C4B9EEA38DD14C3A3ECDEF0A99C1E564CBE66DFE008250154E7839B0B75228FE8DEBCC4CA330E6AEBC1ABC74070BC9C9C1E26B939C9D916E45E13C
  4 : 0xB9D1555BC9F1BCF7AFEAF3D60E246C6063ACF2572518FDC12CF7CD689A1E8E10D0B43CFB77CF60F898D99B5523C829849BF08CA8A882395554FECE71E618BFB2
  5 : 0x58F94888F6AC2C4CC23BFBCFF6A013BED1EFDF239EE1BF2FEF5F8C4E7443A7D2E7A90636792A1B858DC8A70C1BE077DE99992B67CC67AAF1746652FE9043A249
  6 : 0x27EC091533C4B9EEA38DD14C3A3ECDEF0A99C1E564CBE66DFE008250154E7839B0B75228FE8DEBCC4CA330E6AEBC1ABC74070BC9C9C1E26B939C9D916E45E13C
  7 : 0xD2CAB183E6A0BD48AF28C0B04DADAD16EDC21466FB1B8380546147399C42EF82F91A91F2E9D80BEF2EE691E298692775B07B4C02A0C69BD9E55D052865C38302
  8 : 0x77BA9278692B14942F7BCB5E447878D56039E5B649649039B0BBBD31B90DC23996F23213169173A4D30E466A2E98A47BFCBD80EA0E2363BF1AB292E1CB5F6C8F
  9 : 0xC2C007228B3DF18F6749EE86058EC4819833A77E6C3AFE053FBBEB1D1C474180AB9AE75E52621A4AD4E92F39234C5FB787F0576B9DCB292997C05ACEDC770DFC
  10: 0x0DFFB841CDC6998F869A6EEF8A29E89FEC5485A6E0F00347A9D50B19B58DE98F6D66B54A8C10721DA8CC2D52CD09B81CD00F9AB266407961621E4E96E13D767A
  11: 0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  12: 0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  13: 0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  14: 0x71571341801195B199E04CF1A624716DC06C0352B1D8C8A57521BD8E252069E9DA04BCB0E0EA566E496FAFA959E25ACCFD47E0129E42FDCA0DBAE2E31539918B
  15: 0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  16: 0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
        22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
          23: 0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

3.2 内核可信特性

3.2.1 概述

龙蜥操作系统内核继承 linux 内核了成熟的可信计算特性，包括可信根设备驱动、内核完整性架构（Integrity Measurement Archtecture,简称 IMA）、内核可信密钥与加密密钥等。

3.2.2 可信根设备驱动

TPM Eventlog

本节简要描述了什么是 TPM Eventlog（即度量事件日志），以及如何将 TPM Eventlog 从预引导固件传递到操作系统。

1） TPM Eventlog 概述

固件（如 BIOS/UEFI 等）维护了一个 TPM Eventlog，每当固件度量新的组件并将度量结果扩展到任何 PCR 寄存器时，该日志都会新增一条记录。按事件类型分隔，并包含了扩展后的新 PCR 值。TPM Eventlog 的主要用于远程证明，用于向挑战者证明平台的可信状态。TPM Eventlog 的主要作用是提供详细的度量对象和度量结果，远程证明过程中通过包含 PCR 内容的可信报告验证度量日志的可信性。

2） UEFI TPM Eventlog

UEFI 提供了查询 TPM Eventlog 的 protocol 服务。在调用 ExitBootServices() 之前，Linux EFI 会将事件日志复制到由内核自定义的配置表中。 需要说明的是，ExitBootServices()生成的度量结果不会出现在最终的度量日志表中。固件提供了最终事件配置表来解决这个问题。在第一次调用 EFI_TCG2_PROTOCOL.GetEventLog()之后，事件被镜像到最终时间配置表中。

vTPM Proxy Driver for linux Container

1） 面向容器的虚拟 TPM 驱动概述

该特性是为每个 Linux 容器提供 TPM 功能，该特性允许程序以与物理系统上 TPM 交互相同的方式与容器中的 TPM 交互。每个容器都有自己独特的、模拟的软件 TPM。

2) 面向容器的虚拟 TPM 驱动的设计

为了使 swTPM 可用于每个容器，容器管理堆栈需要创建一个设备对，该设备对由一个客户端 TPM 字符设备/dev/tpmX (X=0,1,2...)和一个“服务器端”文件描述符组成。在将文件描述符传递给 TPM 仿真器的同时，通过创建具有适当主编号和副编号的字符设备将前者移动到容器中。然后，容器内的软件可以使用字符设备发送 TPM 命令，仿真器将通过文件描述符接收命令，并使用此文件描述符返回响应。

虚拟 TPM 代理驱动程序提供了一个设备/dev/vtpmx，用于使用 ioctl 创建设备对。ioctl 将其作为配置设备的输入标志。例如，这些标志指示 TPM 模拟器是否支持 TPM 1.2 或 TPM 2 功能。ioctl 的结果是生成“服务器端”的文件描述符以及所创建的字符设备的主次编号，以及返回 TPM 字符设备的编号。例如，如果创建了/dev/tpm10，则返回数字(dev_num) 10。 一旦设备被创建，驱动程序将立即尝试与 TPM 通信。驱动程序的所有命令都可以从 ioctl 返回的文件描述符中读取并立即得到响应。

3) **UAPI** - flags for the proxy TPM

```
4)  enum vtpm_proxy_flags
5)  /**
6)  *  常量
7)  *  VTPM_PROXY_FLAG_TPM2
8)  *       the proxy TPM uses TPM 2.0 protocol
9)  **/
```

- parameter structure for the VTPM_PROXY_IOC_NEW_DEV ioctl

```
1.  struct vtpm_proxy_new_dev
2.
3.  //Definition:
4.  struct vtpm_proxy_new_dev {
5.      __u32 flags;
6.      __u32 tpm_num;
7.      __u32 fd;
8.      __u32 major;
```

```
    __u32 minor;
};
/**结构体成员说明
*  flags: flags for the proxy TPM
*  tpm_num: index of the TPM device
*  fd: the file descriptor used by the proxy TPM
*  major: the major number of the TPM device
*  minor: the minor number of the TPM device
**/
```

- handler for the VTPM_PROXY_IOC_NEW_DEV ioctl

```
long vtpmx_ioc_new_dev(struct file *file, unsigned int ioctl, unsigned long arg)

/**函数参数说明
*  struct file *file     /dev/vtpmx
*  unsigned int ioctl    the ioctl number
*  unsigned long arg     pointer to the struct vtpmx_proxy_new_dev
**/

/*函数功能描述
* 创建一个匿名文件，供进程作为 TPM 与客户端进程通信。
* 该函数还将添加一个新的 TPM 设备，通过该设备将数据代理到该 TPM 代理进程。
* 将为调用者提供一个文件描述符，用于与客户端通信，以及 TPM 设备的主要和次要编号。
*/
```

vTPM interface for Xen

介绍

vTPM interface for Xen 的目标是为虚拟客户机操作系统(用 Xen 术语来说就是 DomU)提供 TPM 功能。这允许程序以与物理系统上的 TPM 交互相同的方式与虚拟系统中的 TPM 交互。每个客户机（Guest OS）都有自己独特的模拟 TPM。然而，每个 vTPM 的秘密(密钥，NVRAM 等)都由 vTPM Manager 域管理，该域将秘密密封到物理 TPM。如果创建这些域(管理器、vTPM 和客户机)的过程是可信的，那么 vTPM 子系统将根植于硬件 TPM 的信任链扩展到 Xen 中的虚拟机。vTPM 的每个主要组件都作为一个单独的域实现，提供由 hypervisor 保证的安全分离。vTPM 域在 mini-o 中实现，以减少内存和处理器开销。 这个 mini-os vTPM 子系统是建立在 IBM 和 Intel 公司以前的 vTPM 工作之上的。

设计概述

vTPM 的架构描述如下:

```
+------------------+
|    Linux DomU    | ...
|       |  ^       |
|       v  |       |
|   xen-tpmfront   |
+------------------+
         |  ^
         v  |
+------------------+
| mini-os/tpmback  |
|       |  ^       |
|       v  |       |
|  vtpm-stubdom    | ...
|       |  ^       |
|       v  |       |
| mini-os/tpmfront |
+------------------+
         |  ^
         v  |
+------------------+
| mini-os/tpmback  |
|       |  ^       |
|       v  |       |
| vtpmmgr-stubdom  |
|       |  ^       |
|       v  |       |
| mini-os/tpm_tis  |
+------------------+
         |  ^
         v  |
+------------------+
|   Hardware TPM   |
+------------------+
```

- Linux DomU：配置有 vTPM 的 Linux 客户机可能不止一个。
- xen-tpmfront.ko：Linux 内核虚拟 TPM 前端驱动。该驱动程序提供对基于 linux 的 DomU 的 vTPM 访问。
- mini-os / tpmback：Mini-os TPM 后端驱动程序，提供 Linux 前端驱动与后端驱动对接功能，实现 Linux DomU 与 vTPM 之间的通信。这个驱动程序也被 vtpmmgr-stubdom 用来与 vtpm-stubdom 通信。

- vtpm-stubdom：一个实现 vTPM 的 mini-os 存根域。在系统上运行的 vtpm-stubdom 实例和逻辑 vtpms 之间存在一对一的映射关系。
- mini-os/tpmfront：Mini-os TPM 前端驱动程序。vTPM mini-os 域 vTPM -stubdom 使用该驱动程序与 vtpmmgr-stubdom 通信。该驱动程序也用于与 vTPM 域通信的 mini-os 域，例如 pv-grub。
- vtpmmgr-stubdom：实现 vTPM 管理器的 mini-os 域。只有一个 vTPM 管理器，它应该在机器的整个生命周期中运行。该域规范对系统物理 TPM 的访问，并确保每个 vTPM 的持久状态。
- mini-os / tpm_tis：Mini-os TPM version 1.2 TPM Interface Specification （TIS）驱动程序。vtpmmgr-stubdom 使用这个驱动程序直接与硬件 TPM 对话。通过将硬件内存页映射到 vtpmmgr-stubdom 的方式，方便与硬件 TPM 通信。
- 硬件 TPM:物理 TPM 模块，一般是焊接到主板上的。

与 Xen 集成

在 Xen 4.3 版本中使用 libxl 工具栈添加了对 vTPM 驱动程序的支持。有关设置 vTPM 和 vTPM Manager 存根域的详细信息，请参阅 Xen 文档(docs/misc/vTPM .txt)。一旦存根域开始运行，就会按照与域配置文件中的磁盘或网络设备相同的方式设置 vTPM 设备。 为了使用 IMA 等需要在 initrd 之前加载 TPM 的特性，必须将 xen-tpmfront 驱动程序编译到内核中。如果不使用这些特性，驱动程序可以被编译为一个模块。

Firmware TPM Driver

介绍

fTPM(Firmware TPM)驱动程序用于支持在 ARM 的 TrustZone 环境中实现的 TPM 设备。驱动程序允许程序以与硬件 TPM 交互相同的方式与 fTPM 交互。

设计

该驱动程序充当一个薄层，向固件中实现的 TPM 传递命令。驱动程序本身不包含太多逻辑，更像是固件和内核/用户空间之间的管道。 fTPM 的实现基于以下文件:https://www.microsoft.com/en-us/research/wp-content/uploads/2017/06/ftpm1.pdf

当驱动程序加载时，它将向用户空间提供名为/dev/tpmX 的字符设备，使用户空间能够通过该设备与 fTPM 通信。

3.2.3 IMA

IMA 是目前使用最为广泛的完整性度量架构。它通过在内核中增加模块，使得当运行程序运行、内核模块被挂载和动态链接库被加载时，对用到的代码和关键数据(如配置文件和结构化数据) 进行一次度量，并将度量结果扩展到 TPM 的 PCR10(默认) 中，同时创建并维护一个度量列表(Measurement List，ML) 。当远程挑战者发起挑战时，将度量列表 ML 和用 TPM 签名的 PCR10 中的度量值发送给挑战者，挑战者通过对比度量值和基准值来判断平台是否可信[6]。

介绍

原始的“ima”模板是固定长度的，包含了文件数据的哈希值和路径名。filedata 哈希值限制为 20 字节(md5/sha1)。路径名是一个以空结尾的字符串，限制为 255 个字符。为了克服这些限制并添加额外的文件元数据，它是扩展当前版本的 IMA 是必要的模板。然而，引入这个特性的主要问题是，每次定义一个新的模板，生成和显示函数度量列表将包括处理新格式的代码因此，随着时间的推移，它会显

著增长。通过分离模板的方案解决这个问题。该方案的核心是定义两个新的数据结构:一个模板描述符，确定度量表中应包括哪些信息；一个模板字段，以生成和显示给定类型的数据。 使用这些结构管理模板非常简单。支持一种新的数据类型，开发人员定义字段标识符并实现两个函数 init()和 show()分别用于生成和显示度量条目。定义一个新的模板描述符需要指定模板格式(由字段标识符分隔的字符串)通过“ima_template_fmt” 内核命令行输入 “|” 字符)参数。在内核初始启动时，IMA 初始化所选的模板描述符通过将格式转换为模板字段结构的数组从支撑点的集合中。在初始化步骤之后，IMA 将调用 ima_alloc_init_template()（在补丁中为新模板管理定义了新功能机制），通过使用模板生成新的度量条目描述符通过内核配置或通过 new 引入“ ima_template” 和“ ima_template_fmt” 内核命令行参数。函数 ima[_ascii]_measurements_show()检索每个条目，用于生成该条目并调用 show()的模板描述符方法，用于模板字段结构数组的每个项。

支持的模板字段和描述符

以下简要罗列出支持的模板字段列表（'identifier': description），用于定义新的模板将描述符的标识符添加到格式字符串中:

- ‘d’ ： 事件的摘要（即度量文件的摘要）；用 SHA1 或 MD5 哈希算法计算;
- ‘n’ ： 事件的名称(即文件名)，大小为 255 字节;
- ‘d-ng’ ： 事件的摘要，用任意散列计算算法（字段格式:<hash 算法>:digest）；
- ‘d-ngv2’ ： 与 d-ng 相同，但前缀为” ima” 或” verity” 摘要类型(字段格式:<摘要类型>:<散列算法>:摘要);
- ‘d-modsig’ ： 事件摘要，不包含附加的 modsig;

- ‘n-ng’ :事件的名称，没有大小限制；
- ‘sig’ ： 文件签名，基于文件的/ fsversity 的摘要，或 EVM 便携式签名。
- ‘modsig’ 附加文件签名；
- ‘buf’ ： 用于生成哈希的缓冲区数据，没有大小限制；
- ‘evmsig’ ： EVM 便携签名；
- ‘iuid’ ： 索引节点 UID；
- ‘igid’ ： 索引节点的 GID；
- ‘imode’ ： 索引节点模式；
- ‘xattrnames’ ： xattr 名称列表(以’ ’ | ’ ’ 分隔)，仅当 xattr 为现在；
- ‘xattrlength’ ： xattr 长度列表(u32)，仅当 xattr 存在时；
- ‘xattrvalues’ ： xattr 值的列表。

下面是定义的模板描述符列表：

- “ima” ： 格式为“d|n”；
- “ima-ng”(默认)：它的格式是“d-ng | n-ng ’ ；
- “ima-ngv2” ： 格式为“d-ngv2|n-ng”；
- “image -sig” ： 格式为“d-ng|n-ng|sig”；
- “ima-sigv2” ： 格式为“d-ngv2|n-ng|sig”；
- “ima-buf” ： 格式为“d-ng|n-ng|buf”；
- “ima-modsig” ： 格式为“d-ng|n-ng|sig|d-modsig|modsig”；
- “evm-sig” ： 格式为“ d-ng|n-ng|evmsig|xattrnames|xattrlength |xattrvalues|iuid|igid|imode”。

IMA 使用

要指定用于生成度量条目的模板描述符，请目前支持以下方法：

- 从内核支持的模板描述符中选择一个配置(ima-ng 是默认选择)；

- 指定模板描述符名称从内核命令行通过 ima_template=参数;
- 注册一个新的模板描述符与自定义格式通过内核命令行参数 ima_template_fmt=。

3.2.4 内核可信密钥与加密密钥支持

可信密钥和加密密钥是添加到现有内核密钥环服务中的两种新密钥类型。这两种新类型都是可变长度对称密钥，所有密钥都是在内核中创建，用户空间只能看到、存储和加载加密的 blob。可信密钥需要可信源的可用性以获得更高的安全性，而加密密钥可以在任何系统上使用。为了方便起见，所有用户级 blob 都以十六进制 ASCII 格式显示和加载，并经过完整性验证。

信任源

可信源为可信密钥提供安全来源。信任源是否足够安全取决于其实现的强度和正确性，以及特定用例的威胁环境。由于内核不知道环境是什么，也没有信任度量，因此它依赖于可信密钥的使用者来确定信任源是否足够安全。 内核支持硬件可信根（如 TPM/TCM）作为内核可信密钥的可信源。由于 TPM/TCM 提供的 SRK 机制确保了根密钥不会离开可信根、且提供芯片级安全保障机制和高安全等级的密钥熵源。

将密钥使用与平台完整性状态绑定

基于 TPM/TCM 提供的密钥服务，密钥可以选择性地密封到指定的 PCR 值，并且只有在 PCR 和 blob 可信验证通过的情况下，TPM 才会对密钥进行解封。加载的可信密钥可以使用新的(未来的)PCR 值更新，因此密钥很容易迁移到新的 PCR 值，例如当内核和 initramfs 更新时。同一个密钥在不同的 PCR 值下可以保存多个 blob，因此很容易支持多个 boot。

接口和 API

内核支持 TPM/TCM 访问接口和 API。

3.2.5 内核可信配置参数(KConfig)

内核中有需要可信计算相关的配置参数，下表中列举出部分常用的参数以及解释其含义。

表 3-2-1 内核可信功能配置参数

内核可信配置名称	用途	依赖的 Config(以下省略 CONFIG_开头)
CONFIG_SYSTEM_TRUSTED_KEYS	系统可信 keys, 可以级联	CRYPTO [=y] && SYSTEM_TRUSTED_KEYRING [=y]
CONFIG_SECONDARY_TRUSTED_KEYRING	允许用户从用户空间加入 system trusted key，要求 key 必须是从用户空间加载，且必须由其中一把 key 签过	CRYPTO [=y] && SYSTEM_TRUSTED_KEYRING [=y]
CONFIG_SYSTEM_EXTRA_CERTIFICATE	设置预留区域允许用户手动插入证书而不需要重编内核	CRYPTO [=y] && SYSTEM_TRUSTED_KEYRING [=y]
CONFIG_SYSTEM_EXTRA_CERTIFICATE_SIZE	证书预留区域的大小	CRYPTO [=y] && SYSTEM_EXTRA_CERTIFICATE [=y]
CONFIG_MODULE_SIG	开启后对内核模块的签名进行检查	MODULES [=y]
CONFIG_MODULE_SIG_FORCE	开启后内核会直接拒绝加载签名有问题的内核模块	MODULES [=y] && MODULE_SIG [=y]
CONFIG_INTEGRITY_AUDIT	使能完整性审计支持	INTEGRITY [=y] && AUDIT [=y]

内核可信配置名称	用途	依赖的 Config(以下省略 CONFIG_开头)
CONFIG_IMA	使能 IMA	INTEGRITY [=y]
CONFIG_IMA_WRITE_POLICY	允许对 IMA 策略的多次写入	INTEGRITY [=y] && IMA [=y]
CONFIG_IMA_READ_POLICY	允许读当前 IMA 策略	INTEGRITY [=y] && IMA [=y]
CONFIG_IMA_X509_PATH	IMA x509 证书路径	INTEGRITY [=y] && IMA_LOAD_X509 [=y]
CONFIG_IMA_DEFAULT_HASH	IMA 缺省的 HASH 算法	INTEGRITY [=y] && IMA [=y]
CONFIG_IMA_DEFAULT_TEMPLATE	IMA 缺省的模板	INTEGRITY [=y] && IMA [=y]
CONFIG_IMA_APPRAISE	使能本地度量完整性评估	INTEGRITY [=y] && IMA [=y]

3.3 可信软件栈

3.3.1 可信软件栈概述

TPM 软件栈 (TSS) 是一种软件规范，提供用于访问 TPM 的标准 API。 应用程序开发人员可以使用此软件规范来更好地使用 TPM 和开发基于 TPM 的应用程序。

3.3.2 各种语言的开源 TPM TSS 软件栈现状

TPM 的软件栈非常繁荣，涉及到多个语言以及多个开源项目。以下表格仅列出部分主流开发语言（C/Go/Java/Python/Rust）的主流 TPM TSS 软件栈现状。

表 3-3-1 各语言 TPM TSS 软件栈现状

TSS 项目	主要贡献者/企业	开发语言	对 TPM1.2/TPM 2.0 的支持情况	简介与现状
tpm2-tss	Intel	C	支持 TPM 2.0	基于 TCG 标准实现的 TPM2 软件栈；包括 FAPI、ESAPI、SAPI、MU、TCTI 多层 APIs，在多个主流 OS 发行版本中被集成，用户量多，基于该开源项目开发的 TPM 知名项目有 tpm2-tools 和 tpm2-abrmd 等。
IBM's TPM 2.0 TSS	IBM	C	支持 TPM 1.2 和 TPM 2.0	在功能上等于但在 API 上不兼容 TCG 标准的 ESAPI、SAPI、TCTI 等 API，用户相比于 tpm2-tss 少很多。
go-tpm	Google	Go	支持 TPM 1.2 和 TPM 2.0	Google 开源的 Go 语言的 TSS 软件栈项目，提供 legacy APIs 和与 TPM2.0 APIs 1:1 映射的 Direct APIs，这些 APIs 没有完整实现 TPM 1.2/2.0 的整个规范。
go-tpm2	canonical	Go	支持 TPM 2.0	canonical 实现的用于与 TPM 2.0 交互的 Go 库，没有 google/go-tpm 活跃，且采用更加严格的 LGPL3.0 license。
JSR 321	MIT	Java	支持 TPM 1.2	JSR 321 是基于 Java 的可信计算 API 标准规范，由 TPM/J 和 IAIK/OpenTC 两个 Java TPM 软件栈开源项目共同发起并在 2011 年被 JCP 接受，但因为

TSS 项目	主要贡献者/企业	开发语言	对 TPM1.2/TPM 2.0 的支持情况	简介与现状
				10年来没有更新后又被撤销，且该标准不支持 TPM 2.0。
tpm2-pytss	Intel	Python	支持 TPM 2.0	pytss 在 tpm2-tss 上实现的 wrapper，便于 python 用户较好地使用 tpm2-tss 软件栈开发基于 TPM 2.0 的 Python 应用程序。
rust-tss-esapi	Arm	Rust	支持 TPM 2.0	rust-tss-esapi 是基于 tpm2-tss 的 ESAPI 开发的 wrapper，便于 RUST 用户较好地使用 tpm2-tss 软件栈开发基于 TPM 2.0 的 RUST 应用程序。包括 Keylime 在内的一些TPM知名开源项目都是该项目的使用者。

3.3.3 龙蜥社区在 TPM TSS 软件栈的贡献

龙蜥社区一直坚持 upstream first 原则，在多个 TSS 上游社区积极贡献代码，一共在tpm2-tss等四个仓库贡献并合入30来个patch, 包括多个feature和bugfix等。

表 3-3-2 龙蜥社区在 TPM TSS 软件栈的贡献

开源软件名称	总计 commit 数量	总计修改行数
tpm2-tss	1	-0/+1
tpm2-abrmd	7	-14/+499
tpm2-tools	21	-52/+982
go-tpm-tools	1	-9/+8

3.3.4 龙蜥 Anolis OS 引入的 TPM2 TSS 软件栈及使用指南

通过上文对各个语言的 TPM TSS 软件栈现状的分析，可以看到 TPM 软件栈分为两类：

- 一种是各个语言原生实现的：其中 tpm2-tss 是基于 TCG 标准实现的被广泛使用。
- 一种是基于其它语言实现的 wrapper（tpm2-pytss 和 rust-tss-esapi 均基于 C 语言的 tpm2-tss 封装的 wrapper)。

结合调研结果，龙蜥社区引入

- tpm2-tss 及配套的 tpm2-abrmd 和 tpm2-tools 来满足大部分可信计算用户的需求。
- python-tpm2-pytss 这个基于 tpm2-tss 的软件栈来满足可信计算 Python 用户的需求。

用户可以在对应的 Anolis OS 版本中 yum 安装即可使用。

此外，值得一提的是：

- 海光在龙蜥社区贡献了 tpm2-tss 和 tpm2-tools 的仓库部分组件/库的国密功能，详见 hygon-tpm2-tss 和 hygon-tpm2-tools，这些特性也都集成到 Anolis OS 对应版本的 yum 源中。
- 龙蜥社区也在跟进和探索知名开源项目 keylime，keylime 部分组件依赖于 rust 的 TSS 软件栈 rust-tss-esapi，未来也有计划将 rust-tss-esapi 引入来更好的服务可信计算 Rust 用户。

3.3.5 tpm2-tss 架构、开发接口和开发示例

tpm2-tss 架构

如图 3-3-1 所示 tpm2-tss 包含以下由高到低的几层软件：FAPI，ESAPI，SAPI，TCTI（TPM Command Transmission Interface， TPM 命令传输接口）等，在 tpm2-tss 和 TPM 2.0 之间还有 TAB（TPM Access Broker，TPM 访问代理），RM（Resource Manager），和设备驱动等软件层次。这些软件层次与 tpm2-tss 包含的软件层次的功能如下：

- FAPI: 大多数的用户层引用程序基于 FAPI 开发就可以了，因为 FAPI 实现了 TPM 百分之八十的常用应用场景。使用这一层开发应用就像是使用 JAVA，C#等高级语言开发应用一样方便。FAPI 对应的库为 libtss2-fapi，对应的标准为 TCG Feature API (FAPI) Specification,TCG TSS 2.0 JSON Data Types and Policy Language Specification。
- ESAPI: 往下一层是 ESAPI，它需要你对 TPM 了解很深，它实现了 TPM2 命令的 1:1 映射，但是同时提供了会话管理以及加解密的辅助功能。这有点像使用 C++开发应用程序。ESAPI 对应的库为 libtss-esys , 对应的标准为 TCG TSS 2.0 Enhanced System API (ESAPI) Specification。
- SAPI: 应用程序也可以直接基于 SAPI 这一层，它实现了 TPM2 命令的 1:1 映射，但这需要你对 TPM 了如指掌。这就像是使用 C 语言编写应用程序，而不是用高级语言。它提供了 TPM 的所有功能，但是要想用好它你必须对 TPM 有很深的理解。SAPI 对应的库为 libtss2-sys，对应的标准为 TCG TSS 2.0 System Level API (SAPI) Specification。

- TCTI: TCTI 层用于向 TPM 发送命令并接收 TPM 对命令的响应。应用可以直接通过 TCTI 发送命令的数据流并解析接收到的响应数据流。这就像是使用汇编语言来编写应用程序。它对应的库为 libtss2-tcti-device、libtss2-tcti-tbs 等，对应的标准为 TCG TSS 2.0 TPM Command Transmission Interface (TCTI) API Specification。
- TAB: TAB 这一层主要负责多线程环境下 TPM 资源的同步。也就是说它允许多个线程同时访问 TPM 而不发生冲突。
- RM: 因为 TPM 内部的存储资源非常有限，所以需要一个资源管理器 RM，它的原理于虚拟内存管理类似，它可以将 TPM 对象和会话换进换出 TPM。
- 驱动：最后一层就是设备驱动，它主要是控制通信外设与 TPM 互相传输数据。如果你愿意的话，直接调用设备驱动接口来编写应用程序也是可以的，当然这就像是你用二进制数据编写程序一样。

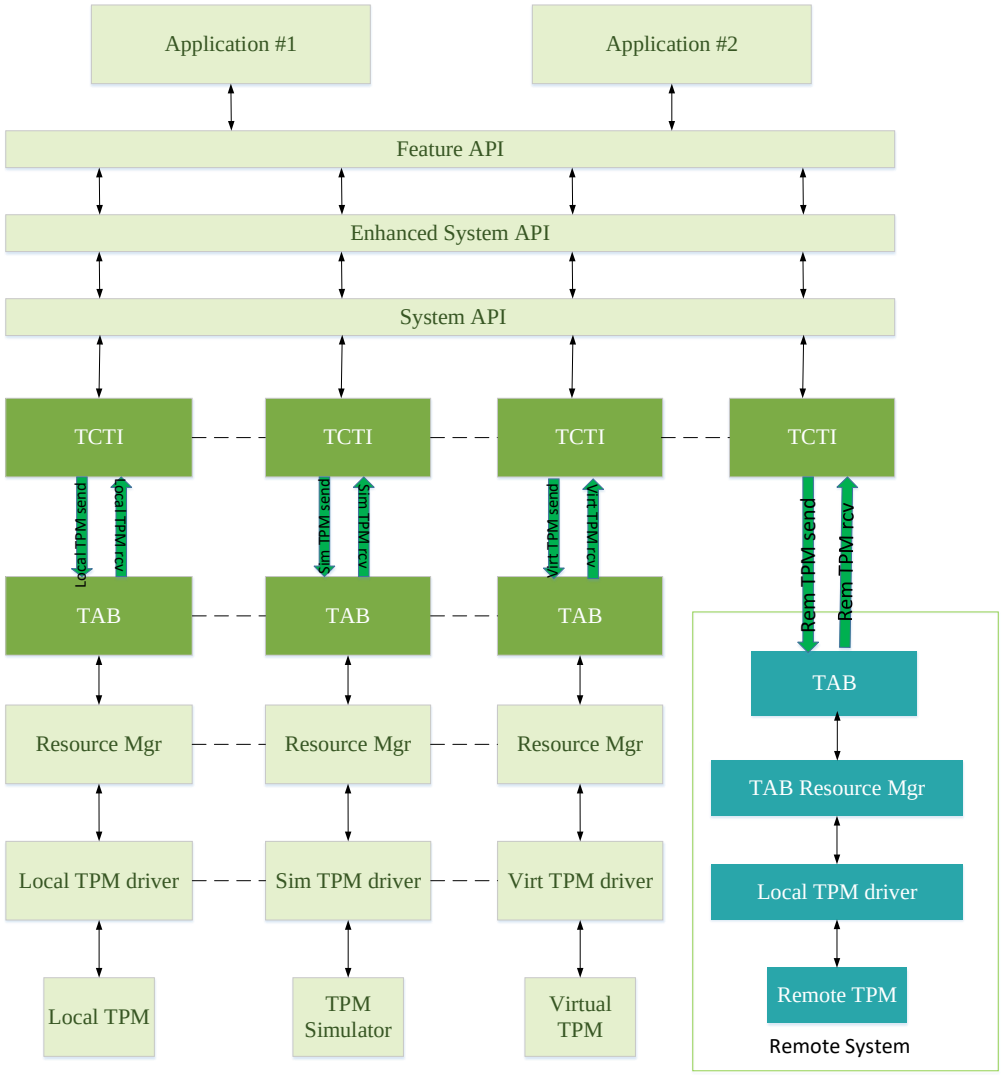


图 3-3-1 tpm2-tss 架构图

tpm2-tss 开发接口

对于开发者和用户而言，tpm2-tss 中使用最多的是 FAPI 和 ESAPI，他们均提供了非常多的 APIs 供开发者使用。tpm2-tss 提供一个文档详细的介绍了 **FAPI** 和 **ESAPI** 中各个 APIs 的用法以及参数的含义，对用户快速理解和使用这些 APIs 非常有帮助。

tpm2-tools 这个开源项目就是使用 ESAPI 进行开发并提供给用户对应的 TPM 2.0 工具。以 ESAPI 为例，ESAPI 的接口分为四类：

- Esys Context ESYS_CONTEXT 相关 APIs：提供一些上下文相关的接口函数，用来初始化和释放上下文、获取底层的 SAPI 和 TCTI 上下文等。
- Esys Tpm Resource ESYS_TR： 负责管理该层 TPM 软件资源相关的 ESAPI。
- Esys TPM Commands： 与 TPM 2.0 命令 1:1 映射的 ESAPI，调用对应的 ESAPI 命令最终会转换为对应的 TPM 2.0 命令。
- Internals of Enhanced System API： 该层内部使用的一些 ESAPI，包轮一些内部类型以及加密相关的 APIs 等。

tpm2-tss 开发示例

tpm2-tools 这个开源项目就是使用 ESAPI 进行开发并提供给用户对应的 TPM 2.0 工具，也是 tpm2-tss 一个好的开发示例。以 tpm2-tools 为例，如果想调用 ESAPI 进行开发，首先参考 tools/tpm2_tool.h 去 include 对应的库，例如：

```
1.  #include <tss2/tss2_esys.h>
```

然后参考 tools/tpm2_tool.c 的 ctx_init 函数去调用 Esys_Initialize 这个 ESAPI 来初始化 ESAPI 的上下文，例如：

```
1.  static ESYS_CONTEXT *ctx_init(TSS2_TCTI_CONTEXT *tcti_ctx) {
2.
3.      ESYS_CONTEXT *esys_ctx;
4.
5.      TSS2_RC rval = Esys_Initialize(&esys_ctx, tcti_ctx, NULL);
6.      if (rval != TPM2_RC_SUCCESS) {
7.          LOG_PERR(Esys_Initialize, rval);
8.          return NULL;
9.      }
10.
11.     return esys_ctx;
12. }
```

接下来调用 ESAPI 的 TPM 命令相关的 API 来与 TPM 进行通信完成对应的操作，比如读取 PCR 值可以参考 lib/tpm2.c 的 tpm2_pcr_read 函数来调用 Esys_PCR_Read 这个 ESAPI 进行读取。

```
tool_rc tpm2_pcr_read(ESYS_CONTEXT *esys_context, ESYS_TR shandle1,
        ESYS_TR shandle2, ESYS_TR shandle3,
        const TPML_PCR_SELECTION *pcr_selection_in, UINT32 *pcr_update_counter,

        TPML_PCR_SELECTION **pcr_selection_out, TPML_DIGEST **pcr_values,
        TPM2B_DIGEST *cp_hash, TPMI_ALG_HASH parameter_hash_algorithm) {

    TSS2_RC rval = TSS2_RC_SUCCESS;
    tool_rc rc = tool_rc_success;
    ……

    rval = Esys_PCR_Read(esys_context, shandle1, shandle2, shandle3,
            pcr_selection_in, pcr_update_counter, pcr_selection_out, pcr_values)
;
    if (rval != TSS2_RC_SUCCESS) {
        LOG_PERR(Esys_PCR_Read, rval);
        return tool_rc_from_tpm(rval);
    }
}
```

最后当我们的所有操作完成后，需要参考 tools/tpm2_tool.c 的 esys_teardown 函数去调用 Esys_Finalize 这个 ESAPI 来销毁 ESAPI 的上下文，比如：

```
static void esys_teardown(ESYS_CONTEXT **esys_context) {

    if (esys_context == NULL)
        return;
    if (*esys_context == NULL)
        return;
    Esys_Finalize(esys_context);
```

3.3.6 龙蜥社区 TSS 软件栈规划

未来龙蜥社区将

- 持续跟进上游社区各个语言 TSS 软件栈的动态并积极参与贡献，同时也会把这些成果引入到 Anolis OS 中。

- 接受社区各个参与方在 TSS 软件栈上的贡献，并以实践文档等方式输出到可信计算 SIG 中。
- 提供更多的实践指南，使用文档等，便于用户更好地使用。

3.4 可信工具集

3.4.1 tpm2-tools 概述

tpm2-tools 是一套基于 TSS2.0（Trusted Software Stack，可信软件栈）接口开发的 TPM2 管理工具集合，可用于操作 TPM2.0 芯片实现密码学、可信存储、完整性验证等功能。

3.4.2 Anolis OS 上 tpm2-tools 实践

Anolis OS 上 tpm2-tools 安装

根据以下命令在 anolis（以 Anolis 8.8 为例）上安装 tpm2-tools

```
1.  yum install tpm2-tools
```

Anolis OS 上 tpm2-tools 使用

tpm2-tools 工具可根据运行环境是否安装 tpm2-abrmd 工具决定 tcti（Transmission Interface）类型，也可以通过-tcti 参数修改 tcti 类型。 以 tpm2_startup 命令为例，未安装 tpm2-abrmd，tpm2_startup 通过 device tcti 与/dev/tpm0 设备直接通信。

```
1.  tpm2_startup -v
2.  tool="tpm2_startup" version="" tctis="libtss2-tctildr" tcti-default=tcti-
    device
```

安装 tpm2-abrmd 后，tpm2_startup 默认通过 tabrmd tcti 与 tpm2-abrmd 服务进程通信，由 tpm2-abrmd 负责与/dev/tpm0 设备进行信息交互。

```
tpm2_startup -v
tool="tpm2_startup" version="" tctis="libtss2-tctildr" tcti-default=tcti-abrmd
```

TPM2.0 基本功能

tpm2_startup 工具可执行 TPM2_CC_Startup 命令使能 TPM2.0 芯片。

```
tpm2_startup  -V #执行 TPM2_SU_STATE 类型的 startup
INFO on line: "54" in file: "tools/tpm2_startup.c": \
Sending TPM_Startup command with type: TPM2_SU_STATE

tpm2_startup -c -V #执行 TPM2_SU_CLEAR 类型的 startup
INFO on line: "54" in file: "tools/tpm2_startup.c": \
Sending TPM_Startup command with type: TPM2_SU_CLEAR
```

tpm2_getcap 工具可执行 TPM2_CC_GetCapability 命令获取 TPM2.0 芯片信息。

```
tpm2_getcap algorithms -V #获取 TPM2.0 芯片支持的算法信息
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x0, property: 0x1
rsa:
  value:      0x1
  asymmetric: 1
  symmetric:  0
  hash:       0
  object:     1
  reserved:   0x0
  signing:    0
  encrypting: 0
  method:     0
.....

tpm2_getcap commands -V #获取 TPM2.0 芯片支持的命令码
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x2, property: 0x11f
TPM2_CC_NV_UndefineSpaceSpecial:
  value: 0x440011F
  commandIndex: 0x11f
  reserved1:    0x0
  nv:           1
  extensive:    0
  flushed:      0
  cHandles:     0x2
  rHandle:      0
  V:            0
  Res:          0x0
......
```

```
31.
32. tpm2_getcap properties-fixed -V #获取 TPM2.0 芯片固定属性信息
33. INFO on line: "44" in file: "lib/tpm2_capability.c": \
34. GetCapability: capability: 0x6, property: 0x100
35. TPM2_PT_FAMILY_INDICATOR:
36.   raw: 0x322E3000
37.   value: "2.0"
38. TPM2_PT_LEVEL:
39.   raw: 0
40. TPM2_PT_REVISION:
41.   value: 1.16
42. TPM2_PT_DAY_OF_YEAR:
43.   raw: 0xF
44. TPM2_PT_YEAR:
45.   raw: 0x7E0
46. TPM2_PT_MANUFACTURER:
47.   raw: 0x564D5700
48.   value: "VMW"
49. .....
50.
51. tpm2_getcap ecc-curves -V #获取 TPM2.0 芯片支持的椭圆曲线信息
52. INFO on line: "44" in file: "lib/tpm2_capability.c": \
53. GetCapability: capability: 0x8, property: 0x1
54. TPM2_ECC_NIST_P192: 0x1
55. TPM2_ECC_NIST_P224: 0x2
56. TPM2_ECC_NIST_P256: 0x3
57. TPM2_ECC_NIST_P384: 0x4
58. TPM2_ECC_BN_P256: 0x10
59. ......
60.
61. tpm2_getcap handles-nv-index -V #获取已定义的 NV 空间句柄
62. INFO on line: "44" in file: "lib/tpm2_capability.c": \
63. GetCapability: capability: 0x1, property: 0x1000000
64. - 0x1691D65
65. - 0x1C00002
66. - 0x1C0000A
67.
68. tpm2_getcap handles-transient -V #获取暂存对象句柄
69. INFO on line: "44" in file: "lib/tpm2_capability.c": \
70. GetCapability: capability: 0x1, property: 0x80000000
71. - 0x80000000
72. - 0x80000001
```

TPM2.0 密码学功能

TPM2.0 密钥管理采用加密存储的方式，每一密钥都有父密钥，密钥导出 TPM2.0 芯片时，都被父密钥加密保护，导入 TPM2.0 芯片时又父密钥解密恢复。在加密存储体系中，存在一个根密钥（又称为 PrimaryObject），该密钥无法导出到

TPM2.0 芯片外。TPM2.0 中有三个独立的特权域（Hierarchy），每一特权域都可创建根密钥。

密钥创建

1）创建根密钥

```
tpm2_createprimary -C o -G rsa -c rsaprimary.ctx -V #在 TPM_RH_Owner Hierary 创建 RSA 算法的
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x5, property: 0x0
name-alg:
  value: sha256
  raw: 0xb
attributes:
  value: fixedtpm|fixedparent|sensitivedataorigin|userwithauth \
  |restricted|decrypt
  raw: 0x30072
type:
  value: rsa
  raw: 0x1
exponent: 0x0
bits: 2048
scheme:
  value: null
  raw: 0x10
scheme-halg:
  value: (null)
  raw: 0x0
sym-alg:
  value: aes
  raw: 0x6
sym-mode:
  value: cfb
  raw: 0x43
sym-keybits: 128
rsa: b7a9f512d495edc54b0fae7a76c8f72a3708f0de4d6a6a08a73547c4d \
f6fddb15e5bf9a94fb5a63ecdeb62e18138d93be4d4522ac12a091b354bab5 \
e4e36dde30b17ae4e84bf5d72a5447f2bfb3e6bc53b9ba847d85c0ec016935 \
4e301dbd9d83ba45a43747d55b54152639786741116da666bfa2fa583e317f \
d1757309a1904c933fae6e92502a01b72bc3f46cc7665852b1a93d3b3344e9 \
5aa254ba4f7d9345916648a7a667a5ae275894a2789b46dff6a26cc8dc4cd8 \
3e848ac7e23a2fa7a0d2091eacb1cd40851eb0bdccb7ebdd1ad8057d1fbc1c \
be54ceacba3e4a90157cfa53adf22f88a7c730b4b1584dff596c62f88ade2a \
8a7c9d67f36f6db169b4f
INFO on line: "190" in file: "lib/files.c": \
Save TPMS_CONTEXT->savedHandle: 0x80000000
```

2）创建密钥

```
tpm2_create -C rsaprimary.ctx -G rsa -u rsa.public \
-r rsa.private -V #以上一步创建的 PrimaryObject 为父密钥，\
创建 RSA 算法的密钥
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x5, property: 0x0
INFO on line: "362" in file: "lib/files.c": \
Assuming tpm context file
INFO on line: "293" in file: "lib/files.c": \
load: TPMS_CONTEXT->savedHandle: 0x80000000
name-alg:
  value: sha256
  raw: 0xb
attributes:
  value: fixedtpm|fixedparent|sensitivedataorigin|userwithauth \
  |decrypt|sign
  raw: 0x60072
type:
  value: rsa
  raw: 0x1
exponent: 0x0
bits: 2048
scheme:
  value: null
  raw: 0x10
scheme-halg:
  value: (null)
  raw: 0x0
sym-alg:
  value: null
  raw: 0x10
sym-mode:
  value: (null)
  raw: 0x0
sym-keybits: 0
rsa: d01e9a0f80a79c7248b29e66535a16c43ff0ad70f5f6773d048bb6e9178 \
78f91ac53f672091b8103123123bce8603d761e7b39eb12b4a286816068c40c4 \
af5bd6296bc565913acc69fa5b4485835f1493a180cfb41ec6d18828f195941a \
6446f55794ab8a304e78d2cf04e52d36a98ae94a70f8fa868dcbd8cf58c909df \
684f0dc1f41ba27bcd86097cb8ae0d3cc50d5fba3ea6efd5780a605536f8a60a \
a95350a0db6d639f5c25732ed4ab122df37d258d6786e0fbb123fc18eab71ed4 \
21c9200b1ebfc47ab5ab0e12a3566fcac5e97b1343ab022bf6ba8a94a1c4b795 \
46208806e3561d405bfdcbd7b2e7205a3fc73ed8e54cac847d32a06f0aec291e \
fb27f
```

注：由于 TPM2.0 芯片中存储空间有限，并不无限加载密钥，tpm2-tools 在管理密钥方面，会将生成的密钥通过 TPM2_CC_ContextSave 将密钥信息导出到文件保存，当使用密钥时，先通过 TPM2_CC_ContextLoad 将密钥信息加载至芯片中，再使用该密钥。

RSA 算法加密/解密

```
tpm2_create -C rsaprimary.ctx -G rsa -u rsa.public -r rsa.private #创建 RSA 算法的密钥

tpm2_load -C rsaprimary.ctx -u rsa.public -r rsa.private \
```

```
-c rsa-enc-key.ctx -V #执行 TPM2_CC_Load 命令将创建的 RSA 密钥加 \
载至 TPM 芯片中
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x5, property: 0x0
INFO on line: "362" in file: "lib/files.c": \
Assuming tpm context file
INFO on line: "293" in file: "lib/files.c":  \
load: TPMS_CONTEXT->savedHandle: 0x80000000
name: 000b0b8d6e072c99c31c90856d9758ca1d2068147e028c \
8073914e4a17a85e573fca
INFO on line: "190" in file: "lib/files.c": \
Save TPMS_CONTEXT->savedHandle: 0x80000000

echo 12345 > data.txt #生成明文

tpm2_rsaencrypt -c rsa-enc-key.ctx -o cipher.bin data.txt -V \
#使用 RSA 密钥加密 data.txt 文件，将密文输出到 cipher.bin 文件中
INFO on line: "362" in file: "lib/files.c": \
Assuming tpm context file
INFO on line: "293" in file: "lib/files.c": \
load: TPMS_CONTEXT->savedHandle: 0x80000000

tpm2_rsadecrypt -c rsa-enc-key.ctx -o data-dec.txt cipher.bin -V \
#使用 RSA 密钥解密密文，并将密文输出到 data-dec.txt 文件
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x5, property: 0x0
INFO on line: "362" in file: "lib/files.c": \
Assuming tpm context file
INFO on line: "293" in file: "lib/files.c": \
load: TPMS_CONTEXT->savedHandle: 0x80000000

diff data-dec.txt data.txt #明文与解密后文件对比
```

RSA 算法签名/验签

```
tpm2_create -C rsaprimary.ctx -G rsa -u rsa.public \
-r rsa.private #创建 RSA 算法的密钥

tpm2_load -C rsaprimary.ctx -u rsa.public -r rsa.private \
-c rsa-sign-key.ctx -V #执行 TPM2_CC_Load 命令将创建的 RSA 密钥 \
加载至 TPM 芯片中

echo "rsasign" > rsasigndata.txt #生成签名内容

tpm2_sign -c rsa-sign-key.ctx -o rsa-sig.bin rsasigndata.txt  -V
#使用 RSA 密钥对 rsasigndata.txt 签名，将签名信息写入 rsa-sig.bin 文件
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x5, property: 0x0
INFO on line: "362" in file: "lib/files.c": \
Assuming tpm context file
INFO on line: "293" in file: "lib/files.c": \
load: TPMS_CONTEXT->savedHandle: 0x80000000

tpm2_verifysignature -c rsa-sign-key.ctx -s rsa-sig.bin \
-m rsasigndata.txt #使用 RSA 密钥验签

```

```
echo "rsasign1" > rsasign1data.txt #构建异常数据

tpm2_verifysignature -c rsa-sign-key.ctx -s rsa-sig.bin \
-m rsasign1data.txt -V #对异常数据签名验签
INFO on line: "362" in file: "lib/files.c": \
Assuming tpm context file
INFO on line: "293" in file: "lib/files.c": \
load: TPMS_CONTEXT->savedHandle: 0x80000000
WARNING:esys:src/tss2-
esys/api/Esys_VerifySignature.c:302:Esys_VerifySignature_Finish() \
Received TPM Error
ERROR:esys:src/tss2-
esys/api/Esys_VerifySignature.c:103:Esys_VerifySignature() \
Esys Finish ErrorCode (0x000002db)
ERROR on line: "53" in file: "lib/log.h": \
Esys_VerifySignature(0x2DB) - tpm:parameter(2):\
the signature is not valid
ERROR on line: "259" in file: "tools/tpm2_verifysignature.c": \
Verify signature failed!
ERROR on line: "147" in file: "tools/tpm2_tool.c": \
Unable to run tpm2_verifysignature
```

ECC 算法签名/验签

```
tpm2_create -C rsaprimary.ctx -G ecc -u ecc.public -r ecc.private #创建 ECC 算法的密钥

tpm2_load -C rsaprimary.ctx -u ecc.public -r ecc.private -c ecc-sign-key.ctx -V \
#执行 TPM2_CC_Load 命令将创建的 ECC 密钥加载至 TPM 芯 片中

echo "eccsign" > eccsigndata.txt #生成签名内容

tpm2_sign -c ecc-sign-key.ctx -o ecc-sig.bin eccsigndata.txt  -V \
#使用 ECC 密钥对 eccsigndata.txt 签名，将签名信息写入 ecc-sig.bin 文件

tpm2_verifysignature -c ecc-sign-key.ctx -s ecc-sig.bin -m eccsigndata.txt #使用 ECC 密钥验签
```

TPM2.0 存储功能

TPM2.0 芯片内置了 NVRAM（Non-Volatile Random Access Memory，非易失性随机访问存储器），用于存放数据。TPM2.0 芯片 NVRAM 读写需要授权，因此可用于存放敏感数据。TPM2.0 NV 空间需要要先定义才能进行读写操作，使用完毕后要释放已定义的空间。

```
tpm2_nvdefine -C o -s 100 0x01800001 -V
#在 TPM_RH_Owner 特权域中创建 100 字节的存储空间，空间索引为 0x01800001
```

```
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x5, property: 0x0
nv-index: 0x1800001

echo "1234567890" > nv.txt #生成存储数据

tpm2_nvwrite -i nv.txt -C o 0x01800001 -V \
#向 NVRAM 0x01800001 写入数据
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x5, property: 0x0
INFO on line: "80" in file: "tools/tpm2_nvwrite.c": \
The data(size=11) to be written:
INFO on line: "1657" in file: "lib/tpm2.c": \
Success to write NV area at index 0x1800001 offset 0x0.

tpm2_nvread -C o 0x01800001 -V \
#读取 NVRAM 0x01800001 中的内容
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x5, property: 0x0
1234567890

tpm2_nvundefine -C o 0x01800001 -V #释放 NVRAM 0x1800001
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x5, property: 0x0
INFO on line: "1580" in file: "lib/tpm2.c": \
Success to release NV area at index 0x1800001.
```

TPM2.0 PCR 功能

TPM2.0 PCR(Platform Configuration Register, 平台配置寄存器)是 TPM2.0 中与完整性相关的信息存储空间。PCR 的更新方法叫做扩展（Extend），扩展是一种单向的加密操作，保证度量值不被篡改。

```
tpm2_pcrread -V #获取当前 PCR 中的内容
sha1:
  0 : 0xA660D212EE691C9295BBEA32A78BE89F9F27C5A9
  1 : 0xB2A83B0EBF2F8374299A5B2BDFC31EA955AD7236
  2 : 0xB2A83B0EBF2F8374299A5B2BDFC31EA955AD7236
  3 : 0xB2A83B0EBF2F8374299A5B2BDFC31EA955AD7236
  4 : 0xB0B51368B2865BD0B8B56BFE1CFE8E6177AB2465
  5 : 0x7273E316E323AAFFBFCDE3ED860DD266E0AA17EB
  6 : 0xB2A83B0EBF2F8374299A5B2BDFC31EA955AD7236
  7 : 0xDBAED2EAEFC85D2342AF7E2C7F0AD9188FF215B1
  8 : 0x16992E4CFBD1E29D5D99ADD56FC9AFAB1EFB0595
  9 : 0x3CFED51D0D507E2CDA5C996374BACAB93C6C6A16
  10: 0x327C45F6007C43E7C6D82958EE6D18F890796A02
  11: 0x0000000000000000000000000000000000000000
  12: 0x0000000000000000000000000000000000000000
  13: 0x0000000000000000000000000000000000000000
  14: 0x8DF12380EDE005407EAB81DA4405321E0DA61280
  15: 0x0000000000000000000000000000000000000000
  16: 0x0000000000000000000000000000000000000000
```

```
  17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  23: 0x0000000000000000000000000000000000000000
sha256:
  0 : 0xCD77123A880A51DB10BBA64BEBFF3B0AD20BA4D50F9F8B8A6B341DBD4E02F468
  1 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
  2 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
  3 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
  4 : 0x76A09DC9FB1E61888B36E6C4A45C02A36B2E39FD40925506110F53586560D4B2
  5 : 0xD004B8D98FBEE7E967E9F46F55CDEE79D487FB5793AA5B1F6D7586A11AD9DEE9
  6 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
  7 : 0x592E7099CFF05155224F26EC6F3781975A7B095F151772CB61714E32F59F6DE1
  8 : 0xE8A441D072B34D68432E00F368BCC3113315CF3707475072BAB93E3195B474C9
  9 : 0x93889BC9C705243940156FAFBD8EDB6CF820962BC45E005BBFFC90C516E991B1
  10: 0x7ABD6A1ADAF3FA4AF27CAA9B541BDB79D535CB577C89FDED6BBF953ACB7AF29B
  11: 0x0000000000000000000000000000000000000000000000000000000000000000
  12: 0x0000000000000000000000000000000000000000000000000000000000000000
  13: 0x0000000000000000000000000000000000000000000000000000000000000000
  14: 0xA4DAD77FB3B6CACBD20F556986C5D917F5E322C123AF82D12C5E5B7EF7AE9938
  15: 0x0000000000000000000000000000000000000000000000000000000000000000
  16: 0x0000000000000000000000000000000000000000000000000000000000000000
  17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  23: 0x0000000000000000000000000000000000000000000000000000000000000000
  ......

echo "123" > pcr.txt #生成扩展数据

sha256sum pcr.txt #计算扩展数据 sha256 摘要值
181210f8f9c779c26da1d9b2075bde0127302ee0e3fca38c9a83f5b1dd8e5d3b   \
pcr.txt

tpm2_pcrextend 10:sha256=181210f8f9c779c26da1d9b2075bde0127302ee0e3fca38c9a83f5b
1dd8e5d3b -V  \
#将 pcr.txt 摘要值扩展至 PCR4 SHA-256 Bank 中
```

TPM2.0 死锁功能

TPM2.0 中对象都需要授权访问，使用错误授权访问具有 DA 保护属性的对象（如密钥、没有设置 noDA 属性的 NV 等）会导致死锁计数器加 1，当死锁计数器达到一定数值后，TPM2.0 便拒绝授权访问。 TPM2.0 中与死锁相关的属性有

maxTries(最大允许授权失败次数)、lockoutRecovery（Lockout Hierarchy 死锁恢复时间），recoveryTime(死锁计数器自减一时间间隔)。

```
tpm2_getcap properties-variable -V #获取与 DA 相关的属性
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x6, property: 0x200
......
TPM2_PT_LOCKOUT_COUNTER: 0x0
TPM2_PT_MAX_AUTH_FAIL: 0x3
TPM2_PT_LOCKOUT_INTERVAL: 0x3E8
TPM2_PT_LOCKOUT_RECOVERY: 0x3E8
......

tpm2_dictionarylockout -s -l 300 -t 300 -n 10 -V \
#设定 maxTries 为 10 次，lockoutRecovery 为 300 秒，recoveryTime 为 300 秒
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x5, property: 0x0
INFO on line: "1110" in file: "lib/tpm2.c": \
Setting up Dictionary Lockout parameters.

tpm2_getcap properties-variable -V
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x6, property: 0x200
......
TPM2_PT_LOCKOUT_COUNTER: 0x0
TPM2_PT_MAX_AUTH_FAIL: 0xA
TPM2_PT_LOCKOUT_INTERVAL: 0x12C
TPM2_PT_LOCKOUT_RECOVERY: 0x12C
......

tpm2_dictionarylockout -c -V #重置死锁计数器
INFO on line: "44" in file: "lib/tpm2_capability.c": \
GetCapability: capability: 0x5, property: 0x0
INFO on line: "1099" in file: "lib/tpm2.c": \
Resetting dictionary lockout state.
```

3.5 可信服务引擎

3.5.1 tpm2-tss-engine 概述

tpm2-tss-engine 利用遵循可信计算组织（Trusted Computing Group，TCG）的软件栈——TSS2.0,实现了基于 TPM2 设备的 OpenSSL 密码引擎。tpm2-tss-engine 利用 TSS2.0 中的增强型系统 API（Enhanced System Applicaion Service

Interface， ESAPI）与 TPM2 设备通信。tpm2-tss-engine 支持 RSA 加解密、签名以及 ECDSA 签名功能。

3.5.2 Anolis OS 上 tpm2-tss-engine 实践

龙蜥社区自其可信计算 SIG 成立以来，一直在关注可信计算业界进展和国际OSV厂商的可信计算方案。在完成tpm2-tss-engine 的实践后，龙蜥社区也将自己的使用经验写入到白皮书中。未来，龙蜥社区除了继续加强与 tpm2-tss-engine 项目的交流与贡献外，还将结合自己在国密/自主创新/云计算的积累围绕 tpm2-tss-engine 开展一些国密支持相关的工作，敬请期待。

Anolis OS 上 tpm2-tss-engine 安装

根据以下命令在 anolis（以 Anolis 8.8 为例）上安装 tpm2-tss-engine：

```
yum install git automake libtool autoconf autoconf-archive \
openssl-devel tpm2-tss-devel tpm2-tools make
git clone https://github.com/tpm2-software/tpm2-tss-engine.git
pushd tpm2-tss-engine
./bootstrap
./configure --prefix=/usr
make
make install
popd
```

Anolis OS 上 tpm2-tss-engine 使用

引擎信息

```
openssl engine -t -c tpm2tss
(tpm2tss) TPM2-TSS engine for OpenSSL
 [RSA, RAND]
      [ available ]
```

随机数

基于 TPM2 生成 128 字节随机数：

```
openssl rand -engine tpm2tss -hex 128
engine "tpm2tss" set.
8a1b6a489fcf1b1fd8324e97cd76ff7e52617373fc43f7227145c69163 \
b85bd15bb77375a4d5a69b998c4717e7b4c8b1bdb1f3b0e3936a6f528d \
9c90189c022cfeb94f008e35d54407c89229ef7fa338f9be0670e8d466 \
0aa61afcdb6e54dccd6079a9e2f93f3ce1528aa8124fcbbadd5bc79296 \
23ce2afe5802af2317b27a43
```

RSA 算法功能

创建密钥

tpm2tss-genkey 创建密钥:

```
tpm2tss-genkey -a rsa -s 2048 rsakey #使用 tpm2-tss-genkey 生成 RSA 算法密钥

openssl rsa -engine tpm2tss -inform engine -in rsakey -pubout \
-outform pem -out rsakey.pub #导出密钥公钥
engine "tpm2tss" set.
writing RSA key

cat rsakey.pub #读取公钥信息
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA17VGBbc3y8/+KSKJ5++K
MiGyY2CXpvgiYcajGZon8dEhYYLZ2d53wk6tgs19rHQl89T7h6rG2i5haaLRLTNr
gkxB4/OfK4dneVEtHgEZLbQmiGoI0ke4wCf9Fhyrl pSRV7EUA0NYg86DE654X8Pd
4VsIc2Wb3Lf1MP1/lX/r5gZknyPqBe7NL5BM46m8WHS25tDf+Mg/vHADgWboVGFK
W+YpxYtubShAgOjXHc5lKuMKqG5nnIJkrxr8hgtf0ZXbVYywMt4NmaYV7Bc632ic
SkHk0OPGZ+RMl8YQmIEmXLK9Tu0IVy58dC0wxvi4V2GQ+p75uWF3K+nZmYUXFl+2
IQIDAQAB
-----END PUBLIC KEY-----
```

使用 TPM2 中已有 RSA 算法密钥

```
tpm2_createprimary -C o -G rsa2048 -c rsaprimary.ctx #创建 TPM2 RSA 算法
Primary Object

tpm2_create -C rsaprimary.ctx -G rsa2048 -u rsa.pub -r rsa.pri #创建 TPM2 RSA 密
钥

tpm2_load -C rsaprimary.ctx -u rsa.pub -r rsa.pri -c rsa.ctx #将 RSA 密钥导入 TPM2
芯片

tpm2_evictcontrol -C o -c rsa.ctx #将 RSA 密钥设置为持久对象
persistent-handle: 0x81000000
action: persisted

openssl rsa -engine tpm2tss -inform engine -in 0x81000000 \
-pubout -outform pem -out rsatpmkey.pub \
#导出 TPM2 中持久对象 0x81000000 的公钥
engine "tpm2tss" set.
```

```
Enter password for user key:
writing RSA key

cat rsatpmkey.pub ##读取公钥信息
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAyA02SWyY6ZYyVR28O0R9
oDxxqd/RkxPa5W/4O773VkCRF8lovREKLrVV7pVHts6cxw8yrLc8Pzq5bArOTPh0
9M45Caxo13uhPd8H8p5UDORvrylJT7bJb5hrfJYyXyvd9FeXLqXexbJOSwPf+vcD
yv6OKccNwCK/3s//89aEm1B8xuYU1TXFnfo/sLJ+trUIiqrP3Aug/5gwB52lzTAX
WSCdogcbRL/AG7F2Zkn/56miZSzQ0I/o2Y/AaYrY3Oj0W/lIJmGTDiD5TbJJS3gQ
GdY1Tr3xf1Xsfo6ihJ0KCx2ZNBdtX7PIpErztLlllUCBGPNUt8OSVA+V6eZANv9B
owIDAQAB
-----END PUBLIC KEY-----
```

加密/解密

使用 tpm2tss-genkey 生成的密钥加解密数据：

```
echo 123456 > mydata #创建明文

openssl pkeyutl -pubin -inkey rsakey.pub -in mydata -encrypt \
-out mycipher #使用公钥加密数据

openssl pkeyutl -engine tpm2tss -keyform engine -inkey rsakey \
-decrypt -in mycipher -out mycipher-dec #使用私钥解密数据

diff mydata mycipher-dec #对比原文与解密后的明文
```

使用 TPM2 中已加载密钥加密数据：

```
openssl pkeyutl -pubin -inkey rsatpmkey.pub -in mydata -encrypt \
-out mycipher #使用公钥加密数据

openssl pkeyutl -engine tpm2tss -keyform engine -inkey 0x81000000 \
-decrypt -in mycipher -out mycipher-dec
#使用 TPM2 中持久对象 0x81000000 私钥解密数据

diff mydata mycipher-dec #对比原文与解密后的明文
```

签名/验签

使用 tpm2tss-genkey 生成的密钥签名验签数据：

```
openssl pkeyutl -engine tpm2tss -keyform engine -inkey rsakey \
-sign -in mydata -out mysig \
#使用 tpm2tss-genkey 生成的密钥 rsakey 签名数据

openssl pkeyutl -pubin -inkey rsakey.pub -verify -in mydata \
-sigfile mysig #使用 rsakey 的公钥验签
```

```
7.  Signature Verified Successfully
```

使用 TPM2 中已加载密钥签名验签：

```
1.  openssl pkeyutl -engine tpm2tss -keyform engine -inkey 0x81000000 \
2.  -sign -in mydata -out mysig #使用 TPM2 中持久对象 0x81000000 签名数据
3.
4.  openssl pkeyutl -pubin -inkey rsatpmkey.pub -verify -in mydata \
5.  -sigfile mysig #使用 TPM2 中持久对象 0x81000000 的公钥验证签名
```

ECC 算法功能

创建密钥

tpm2tss-genkey 创建密钥：

```
1.  tpm2tss-genkey -a ecdsa ecckey ##使用 tpm2-tss-genkey 生成 ECC 算法密钥，默认椭圆曲
    线为 nist_p256
2.
3.  openssl ec -engine tpm2tss -inform engine -in ecckey -pubout \
4.  -outform pem -out ecckey.pub #导出 ECC 密钥公钥
5.
6.  cat ecckey.pub #读取公钥信息
7.  -----BEGIN PUBLIC KEY-----
8.  MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEwCUr6W94NwjHOQVoTdWQfxwXQ/qD
9.  tuy2ZtDVL6yKkqnEJJZ0insTH+uJyeM0o3qeuKuzmlY+Qh053okXoA8t9w==
10. -----END PUBLIC KEY-----
```

使用 TPM2 中已有 ECC 算法密钥：

```
1.  tpm2_createprimary -C o -G ecc -c eccprimary.ctx \
2.  #创建 TPM2 ECC 算法 Primary Object
3.
4.  tpm2_create -C eccprimary.ctx -G ecc -u ecc.pub -r ecc.pri \
5.  #创建 TPM2 ECC 密钥
6.
7.  tpm2_load -C eccprimary.ctx -u ecc.pub -r ecc.pri -c ecc.ctx
8.  #将 ECC 密钥导入 TPM2 芯片
9.
10. tpm2_evictcontrol -C o -c ecc.ctx #将 ECC 密钥设置为持久对象
11. persistent-handle: 0x81000001
12. action: persisted
13.
14. openssl ec -engine tpm2tss -inform engine -in 0x81000001 -pubout \
15. -outform pem -out ecctpmkey.pub #导出 TPM2 中持久对象 0x81000000 的公钥
16. engine "tpm2tss" set.
17. read EC key
18. Enter password for user key:
19. writing EC key
```

```
20.
21. cat ecctpmkey.pub ##读取公钥信息
22. -----BEGIN PUBLIC KEY-----
23. MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEE16KNIBp/Ca7A3U38AKXcRh+Ji0O
24. dSzdfR/+ogYfN/NjvlW18IhKZg0rO2PdIsS2V5neCnffzKwRiVK0CP/Xvw==
25. -----END PUBLIC KEY-----
```

ECDSA 密钥签名/验签

使用 tpm2tss-genkey 生成的密钥签名验签数据：

```
1.  echo 1234567890 > mydata #创建被签名信息
2.
3.  openssl dgst -sha256 -out mydata.sha256 -binary mydata
4.  #创建被签名信息的摘要值
5.
6.  openssl pkeyutl -engine tpm2tss -keyform engine -inkey ecckey -sign -
    in mydata.sha256 -out mysig \
7.  #使用 tpm2tss-genkey 生成的密钥 ecckey 签名数据
8.
9.  openssl pkeyutl -engine tpm2tss -keyform engine -inkey ecckey -verify \
10. -in mydata.sha256 -sigfile mysig
11. #使用 tpm2tss-genkey 生成的密钥 ecckey 验证签名数据
12. engine "tpm2tss" set.
13. Signature Verified Successfully
```

TPM2 中已加载密钥签名验签：

```
1.  openssl pkeyutl -engine tpm2tss -keyform engine -inkey 0x81000001 \
2.  -sign -in mydata.sha256 -out mysig #使用 TPM2 中持久对象 0x81000001 签名数据
3.
4.  openssl pkeyutl -engine tpm2tss -keyform engine -inkey 0x81000001 \
5.  -verify -in mydata.sha256 -sigfile mysig
6.  engine "tpm2tss" set.
7.  Enter password for user key:
8.  Signature Verified Successfully
```

X509 证书功能

自签名证书

使用 TPM2 中密钥生成自签名证书

```
1.  tpm2_createprimary -C o -G rsa2048 -c rsaprimary.ctx #创建 TPM2 RSA 算法
    Primary Object
2.
3.  tpm2_create -C rsaprimary.ctx -G rsa2048 -u rsa.pub -r rsa.pri
4.  #创建 TPM2 RSA 密钥
```

```
5.
6.  tpm2_load -C rsaprimary.ctx -u rsa.pub -r rsa.pri -c rsa.ctx #将 RSA 密钥导入 TPM2
    芯片
7.
8.  tpm2_evictcontrol -C o -c rsa.ctx #将 RSA 密钥设置为持久对象
9.
10. persistent-handle: 0x81000000
11. action: persisted
12.
13. openssl req -new -x509 -engine tpm2tss -keyform engine -key 0x81000000 \
14. -out rsa.crt#使用 TPM2 芯片中持久对象 0x81000000 生成自签名证书
15. engine "tpm2tss" set.
16. Enter password for user key:
17. You are about to be asked to enter information that will be incorporated
18. into your certificate request.
19. What you are about to enter is what is called a Distinguished Name or a DN.
20. There are quite a few fields but you can leave some blank
21. For some fields there will be a default value,
22. If you enter '.', the field will be left blank.
23. -----
24. Country Name (2 letter code) [XX]:CN
25. State or Province Name (full name) []:Shandong
26. Locality Name (eg, city) [Default City]:Jinan
27. Organization Name (eg, company) [Default Company Ltd]:XX
28. Organizational Unit Name (eg, section) []:Anolis
29. Common Name (eg, your name or your server's hostname) []:TC
30. Email Address []:
31.
32. openssl x509 -in rsa.crt -text #查看证书详细信息
33. Certificate:
34.     Data:
35.         Version: 3 (0x2)
36.         Serial Number:
37.             61:78:a3:3b:05:ec:e3:1a:ed:c0:a6:74:c5:ee:c6:60:22:7f:a5:53
38.         Signature Algorithm: sha256WithRSAEncryption
39.         Issuer: C = CN, ST = Shandong, L = Jinan, O = XX, OU = Anolis, CN = TC
40.         Validity
41.             Not Before: Aug 25 16:48:29 2023 GMT
42.             Not After : Sep 24 16:48:29 2023 GMT
43.         Subject: C = CN, ST = Shandong, L = Jinan, O = XX, OU = Anolis, CN = TC

44.         Subject Public Key Info:
45.             Public Key Algorithm: rsaEncryption
46.                 RSA Public-Key: (2048 bit)
47.                 Modulus:
48.                     00:a8:27:60:bd:00:01:03:7c:d0:b4:4b:5e:44:92:
49.                     75:fa:84:5a:8a:80:ad:17:da:e0:6d:96:e9:4e:6f:
50.                     f6:b9:11:84:80:75:ab:66:2a:06:ce:db:59:8d:1f:
51.                     f9:11:54:ba:45:0a:cb:be:da:39:83:53:c8:9f:39:
52.                     9d:91:a7:37:03:9e:6a:dd:bd:89:86:e1:96:38:ff:
53.                     8d:c5:97:d1:1c:da:16:59:dc:98:c7:48:0b:ed:9f:
54.                     73:3f:b3:ac:8e:89:e2:c3:83:db:53:1d:9c:d3:a7:
55.                     f1:ea:33:97:f2:2c:98:04:a8:b1:e9:61:29:d5:78:
56.                     26:ad:d8:31:2f:d6:c6:c3:cf:87:63:4e:9d:2b:c0:
57.                     d1:67:b9:15:51:8a:4d:a7:46:98:fe:d9:83:10:91:
58.                     96:0e:54:cc:e7:77:05:73:0b:e9:f2:a0:18:b7:e4:
59.                     b9:98:96:90:58:8f:6e:e4:01:e6:7e:78:91:07:df:
60.                     04:2c:59:21:38:7e:05:56:27:2e:bf:af:77:d0:6c:
```

```
                    e6:8c:d9:97:f9:0e:58:65:b0:da:d3:6f:f3:33:e2:
                    c6:40:d3:9d:0c:ba:b6:78:5a:14:54:b1:89:09:9e:
                    5f:d4:86:a0:d0:09:41:fa:67:4c:48:02:96:a8:a5:
                    d5:f2:97:80:02:55:c1:b3:f2:b8:c2:32:82:1c:ed:
                    2a:d9
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                09:34:FF:5E:CE:1F:7E:42:C3:3B:59:DA:A7:74:68:B1:65:C7:4B:28
            X509v3 Authority Key Identifier:
                keyid:09:34:FF:5E:CE:1F:7E:42:C3:3B:59:DA:A7:74:68:B1:65:C7:4B:2
8

            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
         4d:02:e3:18:0f:63:1a:08:66:fb:b4:4a:86:f2:68:26:ce:bd:
         52:ac:e3:f3:95:fd:4f:44:31:f2:ce:40:33:61:a1:5e:59:67:
         76:c6:a8:4e:76:db:85:86:67:e4:ee:4c:fd:73:99:6c:12:21:
         bf:7a:71:b1:b4:ff:1a:ea:5a:f7:eb:3d:57:d7:d6:c7:73:db:
         dd:80:9f:95:ad:24:58:e5:dd:06:0a:47:c4:bc:22:2d:6c:54:
         99:1a:c9:6b:75:7e:a2:27:aa:cb:ab:4b:53:1b:be:33:08:7d:
         99:5d:67:4c:c7:4a:77:82:64:e1:30:3c:9d:17:be:88:a1:64:
         6a:c9:7e:ca:e5:48:f5:a2:cd:0e:8e:c9:9a:21:2c:fb:e4:56:
         ce:b1:cf:82:f4:b1:59:eb:a6:d8:0c:27:11:cb:2e:bf:d0:20:
         cc:d0:75:ef:12:af:34:2d:da:0d:cd:ea:a1:3c:0b:26:0f:0a:
         40:c6:9f:be:da:33:47:db:48:97:f5:5e:3b:4e:dd:3c:f8:d3:
         63:94:be:d4:98:c3:3f:8e:e7:71:85:30:71:1c:d4:0d:11:26:
         4c:ee:69:ce:18:2b:2c:16:8a:b8:02:9b:45:e9:ee:39:96:b4:
         76:93:56:e2:7c:c6:ab:1a:b0:89:c1:47:29:27:34:35:14:be:
         43:0e:92:16
-----BEGIN CERTIFICATE-----
MIIDlzCCAn+gAwIBAgIUYXijOwXs4xrtwKZ0xe7GYCJ/pVMwDQYJKoZIhvcNAQEL
BQAwWzELMAkGA1UEBhMCQ04xETAPBgNVBAgMCFNoYW5kb25nMQ4wDAYDVQQHDAVK
aW5hbjELMAkGA1UECgwCWFgxDzANBgNVBAsMBkFub2xpczELMAkGA1UEAwwCVEMw
HhcNMjMwODI1MTY0ODI5WhcNMjMwOTI0MTY0ODI5WjBbMQswCQYDVQQGEwJDTjER
MA8GA1UECAwIU2hhbmRvbmcxDjAMBgNVBAcMBUppbmFuMQswCQYDVQQKDAJYWDEP
MA0GA1UECwwGQW5vbGlzMQswCQYDVQQDDAJUQzCCASIwDQYJKoZIhvcNAQEBBQAD
ggEPADCCAQoCggEBAKgnYL0AAQN80LRLXkSSdfqEWoqArRfa4G2W6U5v9rkRhIB1
q2YqBs7bWY0f+RFUukUKy77aOYNTyJ85nZGnNwOeat29iYbhljj/jcWX0RzaFlnc
       mMdIC+2fcz+zrI6J4sOD21MdnNOn8eozl/IsmASoselhKdV4Jq3YMS/WxsPPh2NO
       nSvA0We5FVGKTadGmP7ZgxCRlg5UzOd3BXML6fKgGLfkuZiWkFiPbuQB5n54kQff
       BCxZITh+BVYnLr+vd9Bs5ozZl/kOWGWw2tNv8zPixkDTnQy6tnhaFFSxiQmeX9SG
       oNAJQfpnTEgClqil1fKXgAJVwbPyuMIyghztKtkCAwEAAaNTMFEwHQYDVR0OBBYE
       FAk0/17OH35CwztZ2qd0aLFlx0soMB8GA1UdIwQYMBaAFAk0/17OH35CwztZ2qd0
       aLFlx0soMA8GA1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBAE0C4xgP
       YxoIZvu0SobyaCbOvVKs4/OV/U9EMfLOQDNhoV5ZZ3bGqE5224WGZ+TuTP1zmWwS
       Ib96cbG0/xrqWvfrPVfX1sdz292An5WtJFjl3QYKR8S8Ii1sVJkayWt1fqInqsur
       S1MbvjMIfZldZ0zHSneCZOEwPJ0XvoihZGrJfsrlSPWizQ6OyZohLPvkVs6xz4L0
       sVnrptgMJxHLLr/QIMzQde8SrzQt2g3N6qE8CyYPCkDGn77aM0fbSJf1XjtO3Tz4
       02OUvtSYwz+O53GFMHEc1A0RJkzuac4YKywWirgCm0Xp7jmWtHaTVuJ8xqsasInB
       RyknNDUUvkMOkhY=
       -----END CERTIFICATE-----
```

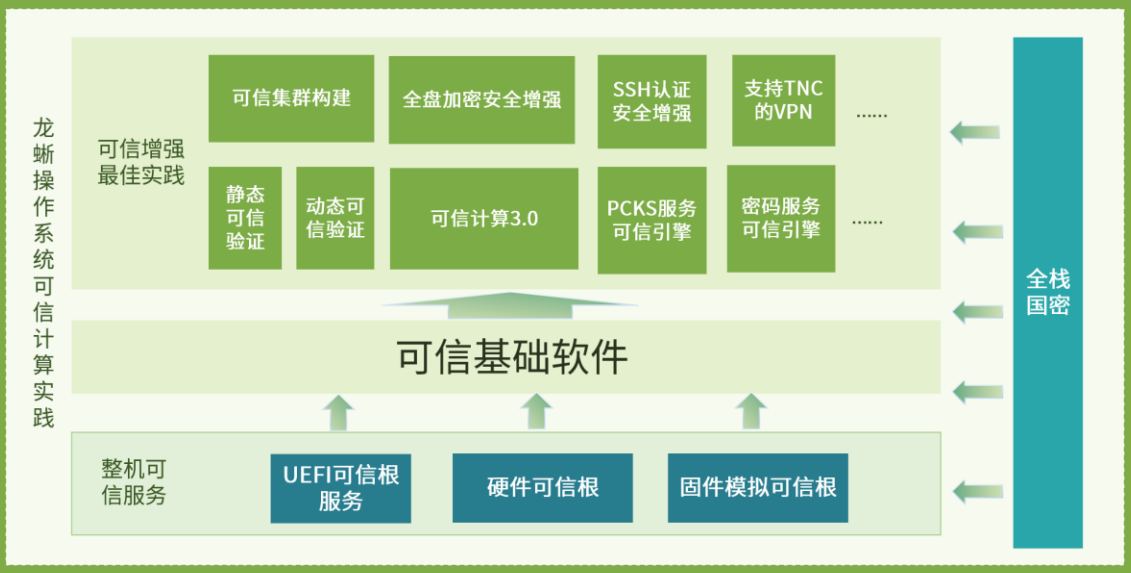
TLS 服务器

通过 tpm2-tss-engine 可使用 TPM2 自签名证书创建 TLS 服务

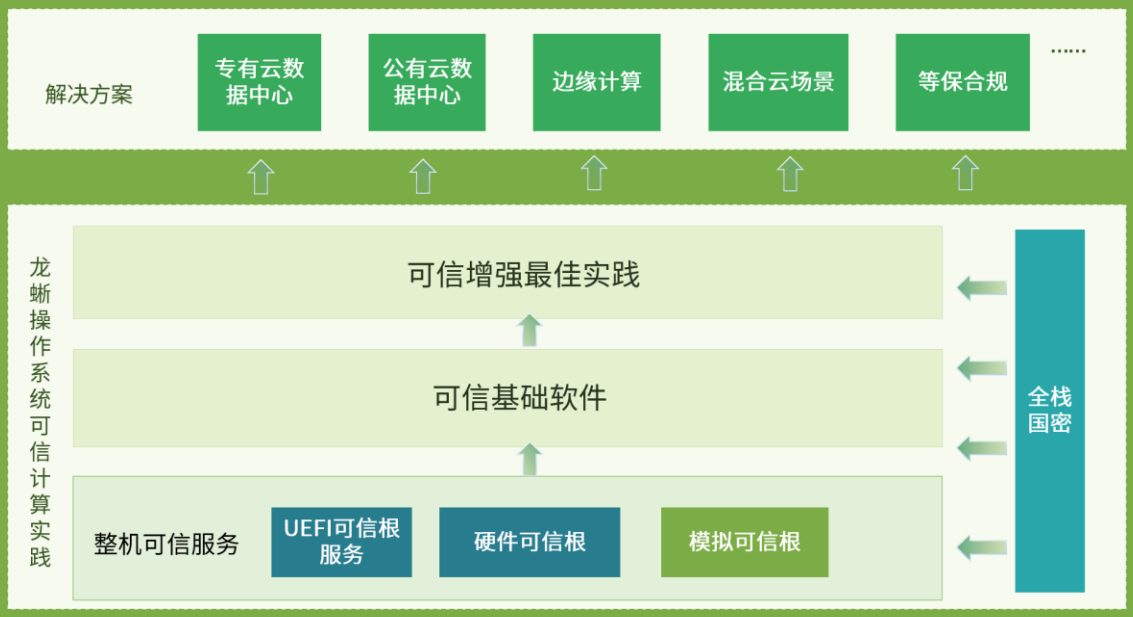
```
openssl s_server -cert rsa.crt -key 0x81000000 -keyform engine \
-engine tpm2tss -accept 8443 #使用 TPM2 自签名证书创建 SSL 服务程序
```

4. 可信计算最佳实践及解决方案

如图所示，可信计算技术作为一种系统级安全增强技术，例如系统可信验证（动态、静态）、系统全盘加密增强、轻量级可信集群构建等，应用得当可以极大的提升操作系统安全能力，这些能力的应用也是可信计算 SIG 探索可信计算最佳实践的重要方向。本节主要主要选编了基于处理器内置可信根服务的可信计算最佳实践，包括基于 keylime 的轻量级可信集群构建实践探索、基于可信根的全盘加密安全增强最佳实践等内容。



在解决方案方面，近年来随着关键信息基础设施建设及合规要求的发展与演变，可信计算面临的场景与挑战也截然不同，例如面向边缘场景的轻量远程证明要求、面向混合云场景的数据安全要求、面向公有云场景的信任优先要求、面向合规建设场景的可信验证及可信计算 3.0 要求等；本节节选了可信计算 3.0 解决方案、基于机密计算的虚拟可信根解决方案等内容。未来，可信计算 SIG 将挑选痛点场景的痛点需求重点开展相关解决方案的挖掘与探索工作。


解决方案
专有云数据中心
公有云数据中心
边缘计算
混合云场景
等保合规
.....
龙蜥操作系统可信计算实践
可信增强最佳实践
可信基础软件
整机可信服务
UEFI可信根服务
硬件可信根
模拟可信根
全栈国密

4.1 海光平台可信计算最佳实践

4.1.1 可信计算整体架构

概述

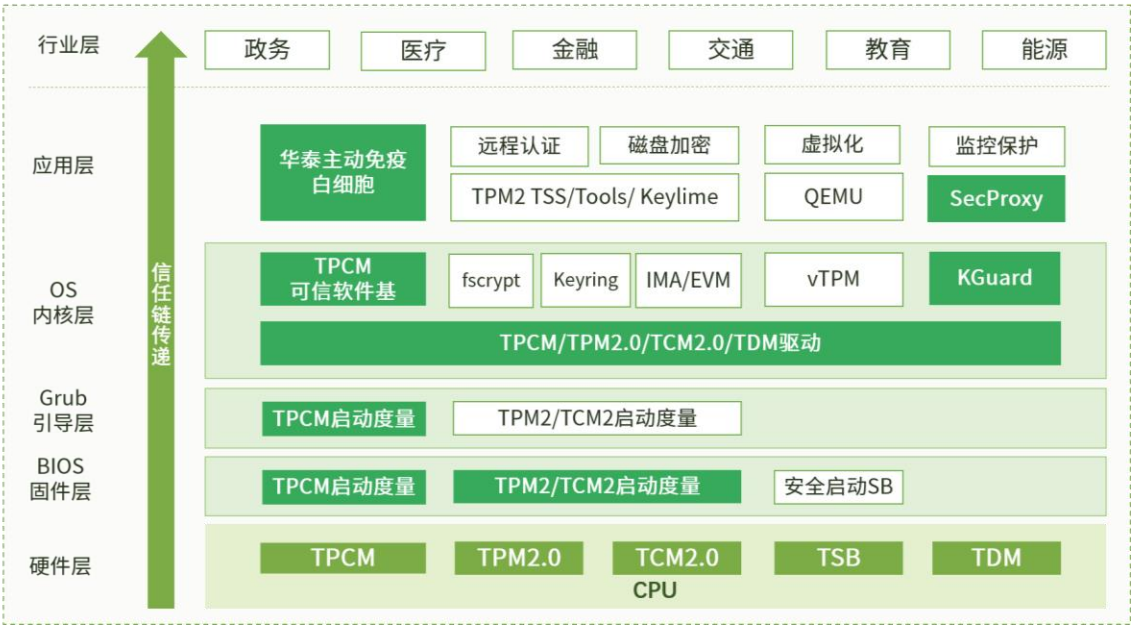


图 4-1-1 可信计算架构

CPU 利用内置安全处理器对可信计算做了相关支持与拓展，在 CPU 内部实现了 TPM2.0 模块（Trusted Platform Module，国际标准）、TCM2.0 模块（Trusted Crypto Module，中国传统标准）、TPCM 模块（Trusted Platform Control Module，中国新兴标准）、TDM 模块（Trusted Dynamic Measuring, 私有）及 TSB 模块（Trusted Secure Boot，私有）五大可信功能模块。利用上述功能模块可以实现可信计算的核心功能，如可信启动、动态度量、可信存储、可信报告等。

可信计算支持

TPM2.0

基于安全处理器以固件的形式实现了 TPM2.0 设备，命令接口符合 TPM2.0 规范。同时，利用内置的密码运算硬件加速引擎 CCP 实现所有密码运算相关操作，克服传统固件实现 TPM 的不足，提高了系统性能，硬件加速引擎全面支持商密标准。

TCM2.0

TCM 是国内传统的可信计算标准，有着广泛的应用。2020 年国密局升级了 TCM 标准，推出了 TCM2.0 规范《GM/T 0012-2020 可信计算 可信密码模块接口规范》。TCM2.0 功能与接口的定义基于 TPM2.0，与 TPM2.0 具有较强的兼容性，TCM2.0 只支持商密标准。与 TPM2.0 一样，TCM2.0 基于 CPU 安全处理器以固件的形式实现，固件以 BIOS PI 的形式发布给 OEM 厂商，因此使用 TCM2.0 需与 OEM 厂商确认 BIOS 已使能该功能支持。

TPCM

TPCM（Trusted Platform Control Module）是国内新兴的可信计算技术，是中国可信计算 3.0 的底层芯片标准，实现可信计算的信任根，由中关村可信计算联盟制定。和 TPM/TCM 相比，TPCM 增加了对系统主动监视和控制的功能，可以实现系统启动时的主动启动度量，及程序运行时的动态度量和监控，进一步增强系统的安全性。

TDM

TDM（Trusted Dynamic Measurement）为基于安全处理器实现的轻量级动态度量，是特有功能。通过 TDM 可以实现对设定内存目标进行持续的周期性度量，

及时发现程序异常，保护程序运行时安全；同时 TDM 通过独有的双重授权保护方式确保非授权用户无法篡改 TDM 内的度量任务设定，极大的增强了模块的安全性。

TSB

平台固件 BIOS 的完整性验证是启动信任链可信的关键和基础，传统的 CPU 由于缺少专门硬件支持无法验证 BIOS 的完整性。TSB（Trusted Secure Boot）是除了上述 TPCM/TPM/TCM 之外另一个独立的由 CPU 硬件验证平台固件 BIOS 完整性的功能，验证基于数字签名。CPU 上电或重启时，TSB 首先验证颁发的 OEM 公钥证书，再用 OEM 公钥证书中的公钥验证 BIOS 固件的 OEM 签名，验证通过后才运行 BIOS 固件，从而保证 BIOS 固件的安全以及后续整个启动信任链的源头安全。

4.1.2 软件编译说明

安装说明

(1)安装 OS 为 Anolis 系列镜像时，功能测试不需要额外安装软件包，可跳过软件编译说明部分直接进行功能测试即可。

(2)安装 OS 镜像为开源版本时，功能测试需要另外安装测试软件包或模块，具体安装说明如下。

tpm2

tpm2 功能测试需要安装相应软件栈，安装包(RPM)的生成及安装脚本已上传龙蜥社区开源仓库，使用如下：

```
1.  $ git clone https://gitee.com/anolis/hygon-devkit.git
2.  $ cd hygon-devkit/tpm/pkg/tpm-1.0.0-20230331/
3.  $ ./install.sh
```

编译并安装完成后按照下文 TPM 功能测试文档的步骤测试即可。

注：tpm2 功能测试还需要安装 grub，具体安装步骤请参考本页面 grub 安装相关说明。

tdm

tdm 驱动已集成到 Anolis OS 5.10 内核。测试 TDM 功能时还需要相应的测试 module，已上传龙蜥社区仓库，安装步骤如下：

```
$ git clone https://gitee.com/anolis/hygon-devkit.git
$ cd hygon-devkit/tdm/pkg/tdm-1.0.0-20230316/
$ make LOCAL_KERDIR=/lib/modules/`uname -r`/build
```

编译通过后可在当前目录下生成 tdm-verify.ko，将该测试 module 拷贝至测试目录下，再按照下文 TDM 功能测试文档的步骤测试即可。

grub

完整的可信启动信任链包含 grub 度量 OS 内核，支持 tpm2 的 grub 版本需 2.04 或以上，同时需将 grub tpm 模块安装进 grub 内核。以 Anolis OS 8.8 为例：

1) 若当前系统 grub 版本（“grub2-install -version”查看）已满足要求，则只需将 tpm 模块安装进 grub 内核，具体步骤如下：

```
$ sudo mkdir /boot/efi/bak
$ sudo cp -fr /boot/efi/EFI /boot/efi/bak/
$ sudo cp -fr /boot/efi/boot /boot/efi/bak/
$ sudo grub2-install --efi-directory=/boot/efi --bootloader-id=anolis --boot-directory=/boot/efi/boot \
--target=x86_64-efi --modules=tpm
$ sudo grub2-mkconfig -o /boot/efi/boot/grub/grub.cfg
```

注：上述/boot/efi/bak 相关步骤（步骤 1~3）是防止安装 grub 出错导致系统无法启动进行的备份操作，下同；不同系统的 grub 安装命令可能会不同，比如 centos 系统，一般使用 grub2-install、grub2-mkconfig 等命令，需要注意区分；grub install 时参数 bootloader-id 可根据系统不同进行替换，当前系统可通过/boot/efi/EFI 目录查看；执行完上述步骤后需要重启。

2) 若当前系统 grub 版本不满足要求，则重新编译安装 grub，具体步骤如下：

```
$ wget https://ftp.gnu.org/gnu/grub/grub-2.04.tar.gz
$ tar -zxvf grub-2.04.tar.gz
$ cd grub-2.04/
$ ./bootstrap
$ ./configure --host=x86_64-linux --target=x86_64 --with-platform=efi
$ make
$ sudo make install
$ sudo cp /etc/default/grub /usr/local/etc/default/
$ sudo mkdir /boot/efi/bak
$ sudo cp -fr /boot/efi/EFI /boot/efi/bak/
$ sudo cp -fr /boot/efi/boot /boot/efi/bak/
$ sudo /usr/local/sbin/grub-install --efi-directory=/boot/efi --bootloader-
id=anolis \
--boot-directory=/boot/efi/boot --target=x86_64-efi --modules=tpm
$ sudo /usr/local/sbin/grub-mkconfig -o /boot/efi/boot/grub/grub.cfg
```

注：bootstrap 在发布版 Grub 里可能没有，没有则不需要运行；拷贝/etc/default/grub 配置文件到/usr/local/etc/default 目录时，如果本地不存在该目录，请创建。

4.1.3 TPM2.0 功能测试

概述

本章节演示了在一台安装 Anolis OS 的服务器上测试 TPM2.0 功能的完整步骤。演示的主要 TPM 应用包括：BIOS/GRUB/Linux 启动度量，商密测试。

说明：

- 发行的 Anolis OS 已包含 TPM2.0 功能在内的安全功能适配支持，安装后用户可以直接测试使用，开源镜像的软件栈安装步骤等请参考 4.1.2 节。
- 支持 tpm2 商密的 tss 和 tools 最低版本分别是 2.3.2-4.0.2 和 4.1.1-5.0.3，如果不满足请更新版本。

表 4-1-1 配置要求

配置	要求
CPU	2 号、3 号
BIOS	PI 版本为 2.1.0.3 或以上，支持 TPM2.0
GRUB	2.04

配置	要求
OpenSSL	openssl-1.1.1k
KERNEL	5.10.134
TPM2-tss	tpm2-tss-2.3.2
TPM2-abrmd	tpm2-abrmd-2.3.3
TPM2-tools	tpm2-tools-4.1.1

本次测试安装的 ISO 镜像名称为 AnolisOS-8.8-x86_64-dvd.iso，具体名称请以发布为准。

BIOS 安装及 TPM 设置

进入 BIOS 设置，在 TCG Trusted Computing 选项下关于 TPM 的默认设置如图 4-1-2 所示，度量使用 SM3 算法，三个 Hierarchy(Platform, Storage, Endorsement)全部使能，如果没特殊需求，使用默认设置即可。BIOS 中开启 TPM 的设置在不同的 BIOS 下可能会有差异，具体请咨询相应的 BIOS 厂商。以下设置仅供参考：

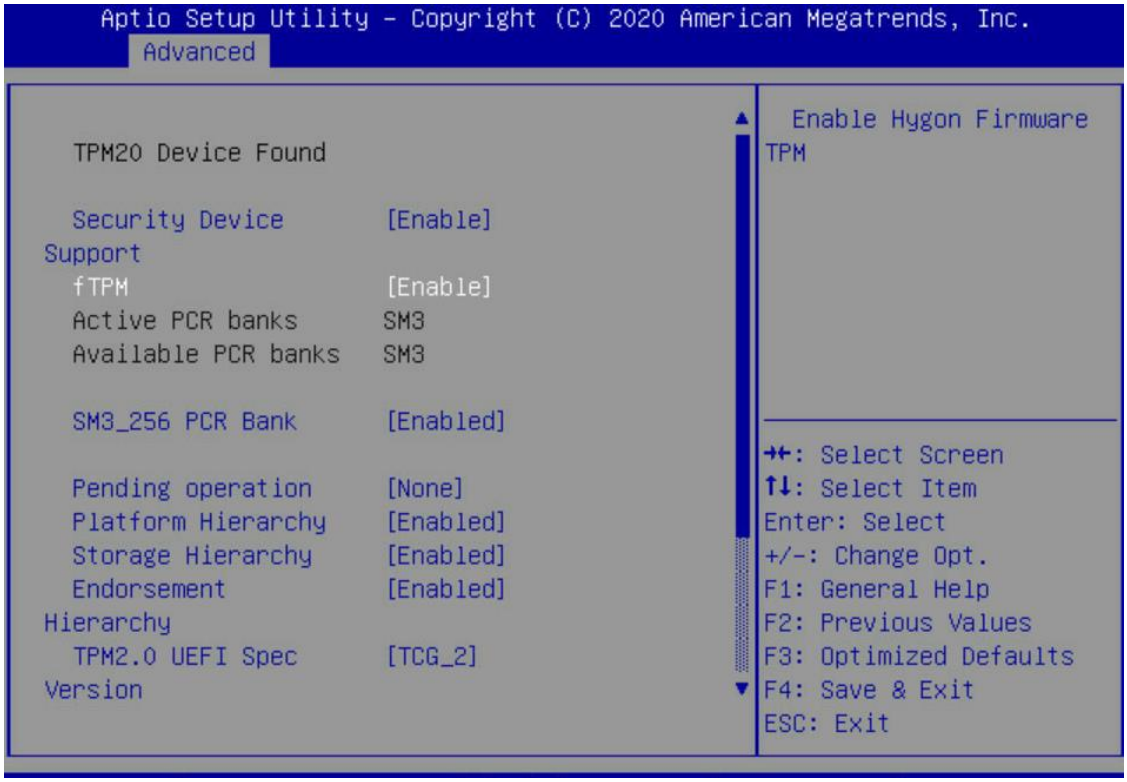


图 4-1-2 BIOS 设置

依赖软件包

tpm2.0 依赖的软件栈或者安装包等都已经集成进 ISO，如果当前环境缺少对应安装包，可直接使用 yum 安装。

需要注意的是 tpm 驱动默认以 ko 模块的方式集成到 kernel 中，如果需要测试 IMA 功能，则需要将 TPM 驱动编译进内核，不能以模块的方式加载。如果不需要使用 IMA 则可以忽略。

配置

安装相应安装包后，在测试 tpm2 功能之前，需要设置如下配置并启动 tpm2-abrmd 服务：

```
1.  $ sudo useradd --system --user-group tss
```

```
$ sudo udevadm control --reload-rules && udevadm trigger
$ sudo pkill -HUP dbus-daemon
$ sudo systemctl daemon-reload
$ sudo ldconfig
$ sudo systemctl enable tpm2-abrmd
$ sudo chown tss:tss /dev/tpm0
$ sudo service tpm2-abrmd start
$ systemctl status tpm2-abrmd.service
```

BIOS/Grub/Linux 内核启动度量

启动度量对 TPM PCR 的使用情况如表 4-1-2：

表 4-1-2 TPM PCR 的使用情况

PCR 编号	度量目标
0	BIOS，包括作为度量根的初始代码
1	BIOS 平台配置
2	OPTION ROM 代码
3	OPTION ROM 配置及数据
4	IPL/Grub 代码
5	IPL/Grub 配置及数据
6	STATE_TRANSITION
7	平台厂商相关控制
8	GRUB 执行的命令字符串，包括配置文件内命令和命令行输入
9	GRUB 打开的文件，包括配置文件，efi 子模块，Linux 内核，Initrd 等
10	Linux IMA 使用

系统起来后使用工具 tpm2_pcrread 读取 PCR。根据 PCR 使用分配，在不改变 BIOS/Grub 配置的情况下，每次系统启动后 PCR 0~9 读值应保持不变，如下：

```
[hygon@localhost ~]$ tpm2_pcrread sm3_256:0,1,2,3,4,5,6,7,8,9,10,11
sm3_256:
  0 : 0x5D25A693796A9D6060834A9FB0AF416E9C9FB4D47A22326BBC45686300B471A3
  1 : 0xFAC21DA05E1F8467972D6ABAF2CBED26FBB81B20A26A2751D32798EE574A7B1F
  2 : 0x0D72B0164E4FA67D6B43D3CB8EAD734737E479767E0D545EFF22C6FE6275B357
  3 : 0x0D72B0164E4FA67D6B43D3CB8EAD734737E479767E0D545EFF22C6FE6275B357
  4 : 0xA2381A4E30198BED49FB10A3D86274930B10D0EE788ED1121D1B83CF814362C9
```

```
    5 : 0xC71BED60766F8B89F6296C076F88E702B0E0474ACB8019AC8B8DB52E66E739ED
    6 : 0x0D72B0164E4FA67D6B43D3CB8EAD734737E479767E0D545EFF22C6FE6275B357
    7 : 0x2304AF3530A51BC03051BA7D3A2BB7B462120DF1B1D13BB55FA0B565831C19F4
    8 : 0xDEA0758622845043428A74802AABB23B53941CD216BA934FBFB5AF4D69FD7905
    9 : 0x2A12748335078EF34ABB0803C293AD390C2D9AB8B6D5FA9086E6A1ED4CD706FD
    10: 0x0000000000000000000000000000000000000000000000000000000000000000
    11: 0x0000000000000000000000000000000000000000000000000000000000000000
```

商密支持的测试

商密测试脚本

包含商密测试的脚本已入库龙蜥仓库，具体见 tpm2-tools 仓库 a8 分支，patch 名称：0001-add-gm-test-case-for-all-commands.patch。

运行测试

命令正确执行的结果如下:

```
[hygon@localhost tests_gm]$ ./test.sh
test_tpm2_activatecredential.sh ... PASSED
test_tpm2_attest.sh ... PASSED
test_tpm2_changeauth.sh ... PASSED
test_tpm2_clock.sh ... PASSED
test_tpm2_encryptdecrypt.sh ... PASSED
test_tpm2_hash.sh ... PASSED
test_tpm2_nv.sh ... PASSED
test_tpm2_pcr.sh ... PASSED
test_tpm2_policy.sh ... PASSED
test_tpm2_random.sh ... PASSED
test_tpm2_selftest.sh ... PASSED
test_tpm2_sign.sh ... PASSED
Tests passed: 12
Tests Failed: 0
```

测试脚本说明

- 每个以 test 开头的脚本是测试 TPM 的一类脚本,比如测试 TPM policy 相关命令的脚本名字为 test_tpm2_policy.sh。
- 脚本中测试命令都是测试商密的命令:

- TPM 算法解析是 object:scheme: symdetail 格式的字符串，比如 tpm2_createprimary -C o -g sm3_256 -G eccsm2:null: sm4128cfb -c /tmp/context;
- 字符串 eccsm2:null:sm4128cfb 是在上面格式的基础上添加商密相关的字符串 sm2，sm4128cfb 以支持商密；
- 在某些命令中需要增加额外的参数支持商密选择，同时保持兼容以前的密码算法，具体命令可参考脚本中 tpm2_certify，tpm2_createak，tpm2_createek，tpm2_createprimary，tpm2_loadexternal，tpm2_nvdefine，tpm2_quote，tpm2_startauthsession 的使用。

4.1.4 TDM 功能测试

概述

如下章节演示了在一台安装 Anolis OS 的服务器上使用 TDM 功能的步骤，演示的主要 TDM 应用包括：TDM 动态保护任务创建运行销毁流程，TDM 动态保护任务更新操作流程，TDM 动态保护防字典攻击流程，TDM 基于虚拟地址创建运行销毁度量任务流程，TDM 度量任务异常触发机器挂机流程，TDM 证书获取与证书链验证流程，TDM 度量报告获取与验证流程，TDM VPCR 获取与重放 VPCR 值验证流程。

注：发行的 Anolis OS 已包含 TDM 功能在内的安全功能适配支持，安装后用户可以直接测试使用，TDM 测试 module 的获取请参照 4.1.2 节。

表 4-1-3 配置要求

配置	要求
CPU	2 号、3 号

配置	要求
BIOS	PI 版本为 2.1.0.3 或以上，支持 TDM
KERNEL	5.10.134
hag	1667 或以上
TDM 固件	1.4 或以上
TDM 驱动	0.7 或以上
TPM2-tools	tpm2-tools-4.1.1

本次测试安装的 ISO 镜像名称为 AnolisOS-8.8-x86_64-dvd.iso，具体名称请以发布为准。

TDM 测试环境配置

TDM 驱动加载

ISO 镜像中的内核已包含 TDM 驱动，机器启动完成后 dmesg 输出下面 log 则表示 TDM 驱动已正常加载。

```
1.  tdm: Thread started for measurement exception handler dispatching...
2.  tdm: TDM driver loaded successfully!
```

使用 hag 查看 TDM 设备信息

注：hag 工具可在龙蜥社区获取，hygon-devkit 仓库 bin 目录下；可将该工具拷贝至测试目录，并添加到 PATH 使用。

输出如下所示，可以看到成功获取 TDM 设备信息：

```
1.  [root@localhost hygon]# hag tdm show_tdm_device
2.  ############################TDM_SHOW_DEVICE#################################
3.  api_major          : 1
4.  api_minor          : 4
5.  buildId            : 1882
6.  task_max           : 100
7.  range_max_per_task: 128
8.  show tdm device successful.
```

```
9.  show_tdm_device command success!
10.
11. [tdm] Command successful!
```

tpm2_tools 配置

tpm2_tools 工具主要用来读取 PCR 的值，VPCR 需要使用到 fTPM，tpm2_tools 工具已默认集成到 ISO 镜像，不需额外配置。

TDM 动态保护任务创建运行销毁流程

该场景主要测试《PSP 动态度量接口规范》中的第七部分“度量任务创建及运行”，实现了度量任务的创建、注册异常回调、启动度量、停止度量、销毁度量任务的完成过程，在验证各个命令正常的情况下同时测试了正常启动度量的流程。

参考用例场景 0 测试主流程逻辑如图 4-1-3 所示，详细细节可阅读参考用例 tdm_verify.c 的代码逻辑：

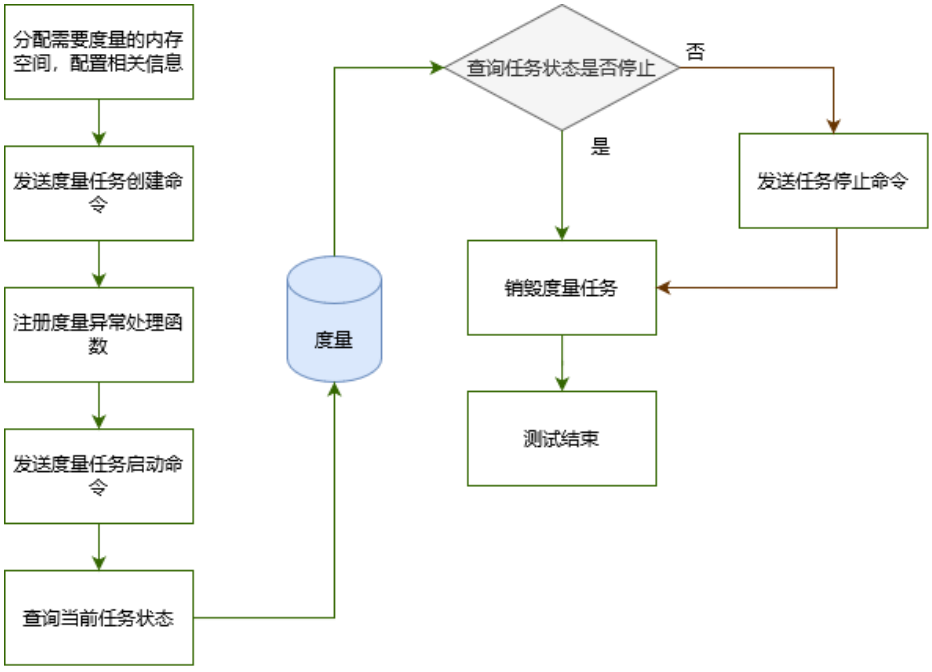


图 4-1-3 TDM 场景 0 测试流程

test_scene 表示测试场景，0 表示正常创建场景测试，1 表示更新操作场景，2 表示防字典攻击场景，3 表示虚拟地址任务场景。

安装本次测试模块：

```
$ sudo insmod tdm-verify.ko test_scene=0
```

移除本次测试模块（相较于以前版本增加了证书链验证与度量报告支持，后续测试需要使用该模块的度量任务，因此在移除模块时才释放度量任务）：

```
$ sudo rmmod tdm-verify.ko
```

查看 Log 信息：

```
$ dmesg
```

输出下面的 log 则表示 TDM 测试场景 0 测试成功，若出现 error 字样，表示该场景测试失败。

```
            ------Victim module: has 3 blocks of data measured by PSP--------
test_scene:0
[23048.936285] Call psp_create_measure_task to request measuring service.
[23048.936286] tdm: TDM: Can't get max_pfn, skip physical address check
[23048.936964] Call psp_register_measure_exception_handler to register measuring
 exception function for task: 0
[23048.937059] Call psp_startstop_measure_task to start measuring for task: 0
[23048.980357] Call psp_create_measure_task to request measuring service.
[23048.980358] tdm: TDM: Can't get max_pfn, skip physical address check
[23048.983682] Call psp_register_measure_exception_handler to register measuring
 exception function for task: 1
[23048.986755] Call psp_startstop_measure_task to start measuring for task: 1
[23049.035503] Call psp_create_measure_task to request measuring service.
[23049.035505] tdm: TDM: Can't get max_pfn, skip physical address check
[23049.039294] Call psp_register_measure_exception_handler to register measuring
 exception function for task: 2
[23049.042308] Call psp_startstop_measure_task to start measuring for task: 2
[23059.357713] Call psp_startstop_measure_task to stop measuring for task: 0
[23059.402299] Call psp_destroy_measure_task to destroy measuring service for ta
sk: 0
[23059.406240] Call psp_startstop_measure_task to stop measuring for task: 1
[23059.451298] Call psp_destroy_measure_task to destroy measuring service for ta
sk: 1
[23059.457734] Call psp_startstop_measure_task to stop measuring for task: 2
```

```
[23059.502293] Call psp_destroy_measure_task to destroy measuring service for task: 2
[23059.502470]
                ------------------------end---------------------------
```

TDM 动态保护任务更新操作流程

该场景主要测试《PSP 动态度量接口规范》中的第八部分“度量任务基准值更新”，实现了对正在正常运行的度量任务的基准值更新，通过更新后基准值与实际度量结果不匹配而出发异常回调，同时验证了更新命令与异常触发回调的流程。

参考用例场景 1 测试主流程如图 4-1-4 所示：

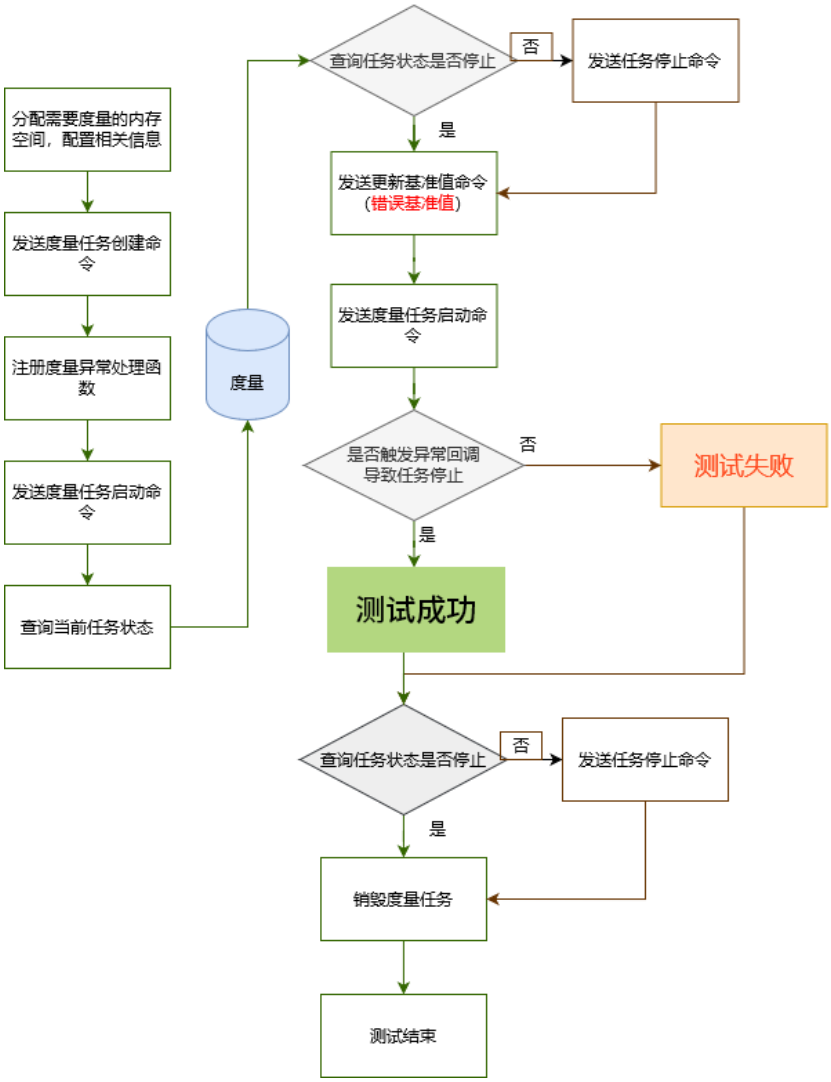


图 4-1-4 TDM 场景 1 测试流程

test_scene 表示测试场景，0 表示正常创建场景测试，1 表示更新操作场景，2 表示防字典攻击场景，3 表示虚拟地址任务场景

安装本次测试模块

```
1.  $ sudo insmod tdm-verify.ko test_scene=1
```

移除本次测试模块（相较于以前版本增加了证书链验证与度量报告支持，后续测试需要使用该模块的度量任务，因此在移除模块时才释放度量任务）

```
$ sudo rmmod tdm-verify.ko
```

查看 Log 信息

```
$ dmesg
```

输出下面的 log 则表示 TDM 测试场景 1 测试成功，默认 demo 测试触发三次异常度量，在内核空间将有三个对应任务 id 的异常函数被触发。

```
          ------Victim module: has 3 blocks of data measured by PSP--------
test_scene:1
[23074.069910] Call psp_create_measure_task to request measuring service.
[23074.069911] tdm: TDM: Can't get max_pfn, skip physical address check
[23074.070566] Call psp_register_measure_exception_handler to register measuring exception function for task: 3
[23074.070661] Call psp_startstop_measure_task to start measuring for task: 3
[23074.112871] Call psp_create_measure_task to request measuring service.
[23074.112872] tdm: TDM: Can't get max_pfn, skip physical address check
[23074.116483] Call psp_register_measure_exception_handler to register measuring exception function for task: 4
[23074.119492] Call psp_startstop_measure_task to start measuring for task: 4
[23074.166990] Call psp_create_measure_task to request measuring service.
[23074.166992] tdm: TDM: Can't get max_pfn, skip physical address check
[23074.170536] Call psp_register_measure_exception_handler to register measuring exception function for task: 5
[23074.173597] Call psp_startstop_measure_task to start measuring for task: 5
[23074.331699] Call psp_startstop_measure_task to stop measuring for task: 3
[23074.375645] Call psp_update_measure_task to update measuring for task: 3
[23074.377384] Call psp_startstop_measure_task to start measuring for task: 3
[23074.394873] tdm: -----Measurement exception handler dispatching thread-----
-
[23074.394874] tdm: Measurement exception received for task 3
[23074.394875] tdm: Step1: Query PSP for task 3 status to confirm the error.
[23074.394876] tdm: Step2: Error confirmed, CALL measurement exception handler.

[23074.394926] Call psp_startstop_measure_task to stop measuring for task: 4
[23074.401037] tdm: Error detected for task 3, action TODO!
[23074.401039] tdm: ----Measurement exception handler----
[23074.401040] ALARM!
[23074.401040] Task:3, corruption detected!
[23074.401041] Please check if it's intended, or your machine may be on danger!

[23074.401041] tdm: Exit measurement exception handler.
[23074.439650] Call psp_update_measure_task to update measuring for task: 4
[23074.440293] Call psp_startstop_measure_task to start measuring for task: 4
```

```
[23074.454929] tdm: -----Measurement exception handler dispatching thread-----
-
[23074.454931] tdm: Measurement exception received for task 4
[23074.454931] tdm: Step1: Query PSP for task 4 status to confirm the error.
[23074.454932] tdm: Step2: Error confirmed, CALL measurement exception handler.

[23074.454953] Call psp_startstop_measure_task to stop measuring for task: 5
[23074.458098] tdm: Error detected for task 4, action TODO!
[23074.458100] tdm: ----Measurement exception handler----
[23074.458100] ALARM!
[23074.458101] Task:4, corruption detected!
[23074.458101] Please check if it's intended, or your machine may be on danger!

[23074.458102] tdm: Exit measurement exception handler.
[23074.502635] Call psp_update_measure_task to update measuring for task: 5
[23074.502728] Call psp_startstop_measure_task to start measuring for task: 5
[23074.517418] tdm: -----Measurement exception handler dispatching thread-----
-
[23074.517420] tdm: Measurement exception received for task 5
[23074.517420] tdm: Step1: Query PSP for task 5 status to confirm the error.
[23074.517421] tdm: Step2: Error confirmed, CALL measurement exception handler.

[23074.517477] tdm: Error detected for task 5, action TODO!
[23074.517478] tdm: ----Measurement exception handler----
[23074.517478] ALARM!
[23074.517479] Task:5, corruption detected!
[23074.517479] Please check if it's intended, or your machine may be on danger!

[23074.517479] tdm: Exit measurement exception handler.
[23076.730176] Call psp_destroy_measure_task to destroy measuring service for ta
sk: 3
[23076.730346] Call psp_destroy_measure_task to destroy measuring service for ta
sk: 4
[23076.730486] Call psp_destroy_measure_task to destroy measuring service for ta
sk: 5
[23076.730651]
                ------------------------end---------------------------
```

TDM 动态保护防字典攻击流程

该场景主要测试《PSP 动态度量接口规范》中的第四部分“防字典攻击”，实现了尝试对正在运行度量任务授权码暴力破解，进而触发 DA 保护启动的逻辑。

参考用例场景 2 测试主流程如图 4-1-5 所示，流程中尝试两次错误的授权，将 DA 保护时间延长至 2^(2-1)=2 秒，并在 DA 保护的 2 秒区间内使用正确的授权码执行命令，由于 DA 保护，将返回在 DA 保护的状态，2 秒保护结束后，使用正确的授权码授权执行命令，将恢复正常，详细逻辑可以参考用例代码。

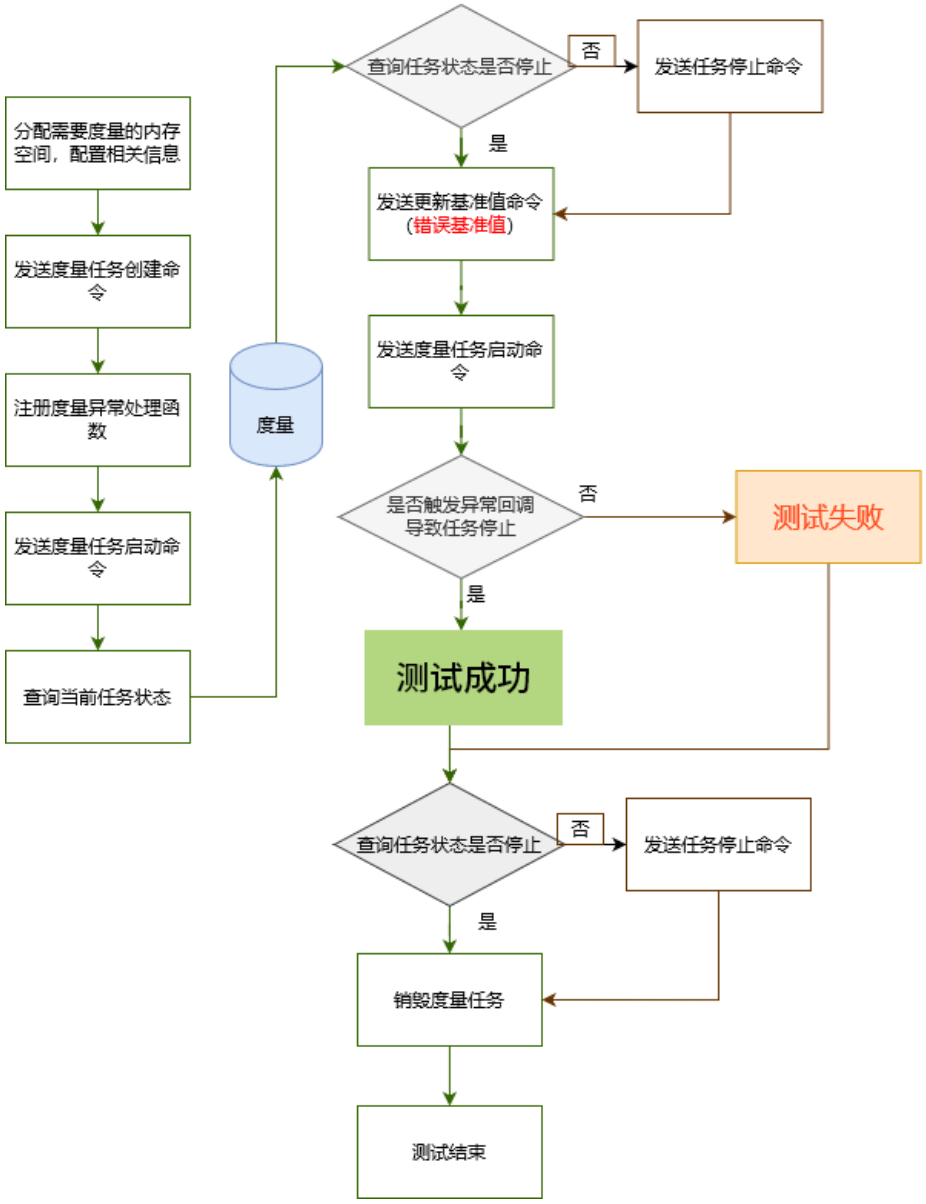


图 4-1-5 TDM 场景 2 测试流程

test_scene 表示测试场景，0 表示正常创建场景测试，1 表示更新操作场景，2 表示防字典攻击场景，3 表示虚拟地址任务场景。

安装本次测试模块：

```
1.  $ sudo insmod tdm-verify.ko test_scene=2
```

移除本次测试模块（相较于以前版本增加了证书链验证与度量报告支持，后续测试需要使用该模块的度量任务，因此在移除模块时才释放度量任务）：

```
$ sudo rmmod tdm-verify.ko
```

查看 Log 信息：

```
$ dmesg
```

输出下面的 log 则表示 TDM 测试场景 2 测试成功，默认 demo 测试三个任务的 DA 攻击，在 DA 保护期间发送命令将拒绝服务返回错误。

```
          ------Victim module: has 3 blocks of data measured by PSP--------
test_scene:2
[23090.387377] Call psp_create_measure_task to request measuring service.
[23090.387378] tdm: TDM: Can't get max_pfn, skip physical address check
[23090.388051] Call psp_register_measure_exception_handler to register measuring
 exception function for task: 6
[23090.388126] Call psp_startstop_measure_task to start measuring for task: 6
[23090.434179] Call psp_create_measure_task to request measuring service.
[23090.434181] tdm: TDM: Can't get max_pfn, skip physical address check
[23090.437755] Call psp_register_measure_exception_handler to register measuring
 exception function for task: 7
[23090.440727] Call psp_startstop_measure_task to start measuring for task: 7
[23090.488312] Call psp_create_measure_task to request measuring service.
[23090.488314] tdm: TDM: Can't get max_pfn, skip physical address check
[23090.491870] Call psp_register_measure_exception_handler to register measuring
 exception function for task: 8
[23090.494848] Call psp_startstop_measure_task to start measuring for task: 8
[23090.541148] Call psp_startstop_measure_task in scene2 to stop measuring for t
ask: 6
[23090.551624] tdm: psp_startstop_measure_task exception error: 0x2
[23091.574741] tdm: psp_startstop_measure_task exception error: 0x2
[23091.577725] tdm: psp_startstop_measure_task exception error: 0x5
[23091.580672] tdm: psp_startstop_measure_task exception error: 0x5
[23093.614989] Call psp_startstop_measure_task in scene2 to stop measuring for t
ask: 7
[23093.626034] tdm: psp_startstop_measure_task exception error: 0x2
[23094.646682] tdm: psp_startstop_measure_task exception error: 0x2
[23094.649850] tdm: psp_startstop_measure_task exception error: 0x5
[23094.649913] tdm: psp_startstop_measure_task exception error: 0x5
[23096.688192] Call psp_startstop_measure_task in scene2 to stop measuring for t
ask: 8
[23096.695727] tdm: psp_startstop_measure_task exception error: 0x2
[23097.720379] tdm: psp_startstop_measure_task exception error: 0x2
[23097.720457] tdm: psp_startstop_measure_task exception error: 0x5
[23097.723599] tdm: psp_startstop_measure_task exception error: 0x5
[23103.104351] Call psp_destroy_measure_task to destroy measuring service for ta
sk: 6
```

```
[23103.104526] Call psp_destroy_measure_task to destroy measuring service for task: 7
[23103.104664] Call psp_destroy_measure_task to destroy measuring service for task: 8
[23103.104823]
                ----------------------end---------------------------
```

4.6. TDM 度量任务通过虚拟地址创建运行销毁流程（1.3 固件版本后支持）

参考用例场景 3 测试主流程逻辑与场景 0 基本相同，主要调整为度量任务创建改用虚拟地址进行创建，详细细节可阅读参考用例 tdm_verify.c 的代码逻辑。

test_scene 表示测试场景，0 表示正常创建场景测试，1 表示更新操作场景，2 表示防字典攻击场景，3 表示虚拟地址任务场景。

安装本次测试模块：

```
$ sudo insmod tdm-verify.ko test_scene=3
```

移除本次测试模块（相较于以前版本增加了证书链验证与度量报告支持，后续测试需要使用该模块的度量任务，因此在移除模块时才释放度量任务）：

```
$ sudo rmmod tdm-verify.ko
```

查看 Log 信息：

```
$ dmesg
```

输出下面的 log 则表示 TDM 测试场景 3 测试成功。

```
          ------Victim module: has 3 blocks of data measured by PSP--------
test_scene:3
[23115.105615] Call psp_create_measure_task to request measuring service.
[23115.106269] Call psp_register_measure_exception_handler to register measuring exception function for task: 9
[23115.106351] Call psp_startstop_measure_task to start measuring for task: 9
[23115.151078] Call psp_create_measure_task to request measuring service.
[23115.154563] Call psp_register_measure_exception_handler to register measuring exception function for task: 10
[23115.157550] Call psp_startstop_measure_task to start measuring for task: 10
[23115.204366] Call psp_create_measure_task to request measuring service.
```

```
[23115.207533] Call psp_register_measure_exception_handler to register measuring exception function for task: 11
[23115.210041] Call psp_startstop_measure_task to start measuring for task: 11
[23117.109442] Call psp_startstop_measure_task to stop measuring for task: 9
[23117.153770] Call psp_destroy_measure_task to destroy measuring service for task: 9
[23117.157152] Call psp_startstop_measure_task to stop measuring for task: 10
[23117.200769] Call psp_destroy_measure_task to destroy measuring service for task: 10
[23117.204428] Call psp_startstop_measure_task to stop measuring for task: 11
[23117.247762] Call psp_destroy_measure_task to destroy measuring service for task: 11
[23117.247932]
                ----------------------end---------------------------
```

TDM 度量任务异常触发机器挂机流程（1.4 固件版本后支持）

参考用例场景 4 测试主流程逻辑与场景 1 基本相同，主要调整为度量任务创建时 flag 配置增加 TASK_EXCEPTION_CRASH 属性配置，当度量异常触发时，机器将挂机，详细细节可阅读参考用例 tdm_verify.c 的代码逻辑。

```
$ cd /lib/modules/`uname -r`/kernel/drivers/crypto/ccp/
```

test_scene 表示测试场景，0 表示正常创建场景测试，1 表示更新操作场景，2 表示防字典攻击场景，3 表示虚拟地址任务场景，4 表示异常挂机场景。

安装本次测试模块：

```
$ sudo insmod tdm-verify.ko test_scene=4
```

该命令运行后，系统触发度量任务异常，若机器此时挂机，表明该机制验证成功，仅可通过 BMC 硬重启或断电重启恢复，用户可根据 verify 的代码逻辑参考该机制的实现，否则该机制验证失败。

TDM AK 证书获取与证书链验证流程（1.2 固件版本后支持）

该场景演示了获取 TDM 的 AK 证书与验证 AK 证书的证书链的例子，通过该场景，可以熟悉并验证 TDM 模块的证书导出接口，同时通过 hag 完成对证书的导出、

解析、验证来实现对整个 TDM 证书系统的了解。hag 提供了证书获取以及从 AK 证书逐步验证整个证书链的过程，用户可以通过使用 hag 工具来熟悉 TDM AK 证书的使用。

基本验证流程如下：

(1) 假设 hag 工具已安装到/usr/bin/目录（具体路径请以安装为准）。

(2) 检查 hag 支持的 TDM 命令。

```
[root@localhost hygon]# hag tdm -help

get_ak_cert         get_tdm_report      get_vPCR_audit      show_tdm_device
parse_ak_cert       verify_ak_cert      parse_tdm_report    verify_tdm_report
parse_vPCR_audit    replay_vPCR_audit

[tdm] Command successful!
```

(3) 可以看到 TDM 支持 get_ak_cert、parse_ak_cert、veriry_ak_cert 三个与证书相关的命令，获取名字为 cert 的 AK 证书。结果如下所示，可以看到成功获取 ak.cert 的证书。

```
[root@localhost hygon]# hag tdm get_ak_cert -out ak.cert
get tdm ak cert successful.
get_ak_cert command success!

[tdm] Command successful!
[root@localhost hygon]# ls -l
总用量 4
-rw-r--r-- 1 root root 448 9 月   1 11:32 ak.cert
```

(4) 解析证书，主要将证书中的版本、chip_id、curve_id、公钥信息、证书签名等信息进行解析显示，方便用户了解证书中的内容。

```
[root@localhost hygon]# hag tdm parse_ak_cert -in ak.cert
##############################TDM_CERT START###################################
version: 10000

chip_id_len:            13
chip_id:
0x4e 0x5a 0x47 0x46 0x47 0x30 0x36 0x31 0x30 0x31 0x38 0x30 0x35
chip_id:                NZGFG06101805

curve_id:               3
```

```
qx:
0xb2 0xcd 0x0c 0x07 0x44 0x61 0x9d 0x97 0x00 0xd0 0xb9 0x72 0x65 0xe3 0x1a 0xba
0x21 0x2d 0x66 0x40 0x51 0xe7 0xf6 0xac 0x2f 0x7d 0xcb 0x0c 0x17 0xd0 0x8b 0xb5

qy:
0x4a 0x4b 0xca 0x88 0xcb 0x1d 0xb1 0x29 0x2f 0x4d 0x50 0xf6 0x5c 0xff 0xa8 0xdd
0x1f 0xcb 0x4c 0x09 0x22 0xf7 0xe9 0x5b 0x9f 0xe9 0xd0 0x8a 0x5d 0x0f 0x45 0xcd

user_id_len:            15
user_id:
0x48 0x59 0x47 0x4f 0x4e 0x2d 0x53 0x53 0x44 0x2d 0x54 0x44 0x4d 0x41 0x4b

sig1_key_usage_id:      0x1004
sig1_r:
0x98 0x3d 0xeb 0x96 0x2e 0x6f 0xb8 0xcf 0xec 0x5a 0x0c 0x5a 0xaf 0xf1 0xb8 0x2d
0xdc 0xaa 0x55 0x77 0x01 0xd0 0x74 0x1a 0x66 0x9e 0x60 0x3d 0xa6 0xf0 0xec 0x16

sig1_s:
0xd5 0x09 0x57 0x7f 0x54 0x30 0x0e 0x8c 0x7c 0xf3 0x34 0x06 0xc4 0xa5 0xd1 0x46
0xaf 0x67 0xbc 0x8d 0xb7 0x19 0xfd 0xb5 0xf1 0xdc 0x54 0x0a 0x41 0xde 0x16 0x51

sig2_key_usage_id:      0x1000
sig2_r:
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

sig2_s:
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

ak cert origin data:
0x00 0x00 0x01 0x00 0x00 0x00 0x0d 0x00 0x4e 0x5a 0x47 0x46 0x47 0x30 0x36 0x31
0x30 0x31 0x38 0x30 0x35 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x01 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x03 0x00 0x00 0x00
0xb2 0xcd 0x0c 0x07 0x44 0x61 0x9d 0x97 0x00 0xd0 0xb9 0x72 0x65 0xe3 0x1a 0xba
0x21 0x2d 0x66 0x40 0x51 0xe7 0xf6 0xac 0x2f 0x7d 0xcb 0x0c 0x17 0xd0 0x8b 0xb5
0x4a 0x4b 0xca 0x88 0xcb 0x1d 0xb1 0x29 0x2f 0x4d 0x50 0xf6 0x5c 0xff 0xa8 0xdd
0x1f 0xcb 0x4c 0x09 0x22 0xf7 0xe9 0x5b 0x9f 0xe9 0xd0 0x8a 0x5d 0x0f 0x45 0xcd
```

```
0x0f 0x00 0x48 0x59 0x47 0x4f 0x4e 0x2d 0x53 0x53 0x44 0x2d 0x54 0x44 0x4d 0x41
0x4b 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x10 0x00 0x00
0x98 0x3d 0xeb 0x96 0x2e 0x6f 0xb8 0xcf 0xec 0x5a 0x0c 0x5a 0xaf 0xf1 0xb8 0x2d
0xdc 0xaa 0x55 0x77 0x01 0xd0 0x74 0x1a 0x66 0x9e 0x60 0x3d 0xa6 0xf0 0xec 0x16
0xd5 0x09 0x57 0x7f 0x54 0x30 0x0e 0x8c 0x7c 0xf3 0x34 0x06 0xc4 0xa5 0xd1 0x46
0xaf 0x67 0xbc 0x8d 0xb7 0x19 0xfd 0xb5 0xf1 0xdc 0x54 0x0a 0x41 0xde 0x16 0x51
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x10 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

###########################TDM_CERT END###################################
parse tdm ak cert successful.
parse_ak_cert command success!

[tdm] Command successful!
```

(5) 验证证书链

```
[root@localhost hygon]# hag tdm verify_ak_cert -in ak.cert
--2023-09-01 11:34:24--  https://cert.hygon.cn/hsk_cek?snumber=NZGFG06101805
正在解析主机 cert.hygon.cn (cert.hygon.cn)... 172.23.18.50
正在连接 cert.hygon.cn (cert.hygon.cn)|172.23.18.50|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度：2916 (2.8K) [binary]
正在保存至: “hsk_cek.cert”

```

```
hsk_cek.cert    100%[=================================================== \
==================================================>]   2.85K  --.-KB/s  用时 0s

2023-09-01 11:34:24 (104 MB/s) - 已保存 “hsk_cek.cert” [2916/2916])

--2023-09-01 11:34:24--  https://cert.hygon.cn/hrk
正在解析主机 cert.hygon.cn (cert.hygon.cn)... 172.23.18.50
正在连接 cert.hygon.cn (cert.hygon.cn)|172.23.18.50|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度：832 [binary]
正在保存至：“hrk.cert”

hrk.cert                                              100%[=================== \
==================================================>]     832  --.-KB/s  用时 0s

2023-09-01 11:34:24 (51.0 MB/s) - 已保存 “hrk.cert” [832/832])

hrk pubkey verify hrk cert successful
hrk pubkey verify hsk cert successful
hsk pubkey verify cek cert successful

[root@localhost hygon]# ls -l
总用量 20
-rw-r--r-- 1 root root  448 9 月   1 11:32 ak.cert
-rw-r--r-- 1 root root 2084 9 月   1 11:34 cek.cert
-rw-r--r-- 1 root root  832 9 月   1 11:34 hrk.cert
-rw-r--r-- 1 root root 2916 9 月   1 11:34 hsk_cek.cert
-rw-r--r-- 1 root root  832 9 月   1 11:34 hsk.cert
```

(6) 可以看到成功验证了 TDM 的证书链，该验证方式主要通过从 AK 证书中获取的 chip_id，到证书服务器下载其对应的 CEK 证书、HSK 证书、HRK 证书，通过从根证书一级一级验证到 AK 证书。

TDM 度量报告获取与使用 AK 证书验证度量报告流程（1.2 固件版本后支持）

该场景演示了获取 TDM 度量报告与使用 AK 证书验证报告的例子，通过该场景，可以熟悉并验证TDM模块的度量报告导出接口，同时通过hag完成对报告的导出、解析、验证来实现对整个TDM 度量报告系统的了解。hag 工具提供了度量报告获取以及使用 AK 证书验证度量报告的过程，用户可以通过使用 hag 工具来熟悉 TDM 度量报告接口的使用。

基本验证流程如下：

(1) TDM 支持 get_tdm_report、parse_tdm_report、verify_tdm_report 三个与报告相关的命令，其中get_tdm_report命令需要三个参数，第一个为报告名字，这里以 report.bin 为例，第二个为报告类型，分别为总结报告（0 对应）与详细报告（1 对应），第三个为用户提供的数据的文件名字，这里以 user_data.bin 为例，第四个为可选的任务号（默认为所有任务）。verify_tdm_report 需要两个参数，第一个为验签的 AK 证书，第二个为被验证的度量报告文件。

(2) 目前最新的 tdm_verify 经过调整后，可以支持 TDM 报告验证，调整为加载 verify 模块创建启动度量任务后不销毁，待到移除该模块时才销毁，因此加载 verify 模块后再调用度量报告获取命令可以获取实际 TDM 中运行的度量报告，这里以获取 tdm_verify 的场景 1 任务更新操作流程中的度量任务的详细报告为例进行说明。

(3) 加载 tdm_verify 的场景 1 模块

```
$ sudo insmod tdm-verify.ko test_scene=1
```

(4) 获取并查看用户提供的随机数文件：

```
$ dd if=/dev/random of=user_data.bin bs=32 count=1
$ hexdump user_data.bin
```

(5) 在 hag 工具的目录下获取名为 report.bin 的详细度量报告，可以看到成功获取 report.bin 的证书。获取度量报告结果如下：

```
[root@localhost hygon]# hag tdm get_tdm_report -report_file report.bin -
report_type 1 \
```

```
-user_data_file user_data.bin
######### Report type    : 1 #########
######### Report task id: 0xffffffff #########
get tdm report successful.
get_tdm_report command success!

[tdm] Command successful!
[root@localhost hygon]# ls -l
总用量 28
-rw-r--r-- 1 root root  448 9月   1 11:32 ak.cert
-rw-r--r-- 1 root root 2084 9月   1 11:34 cek.cert
-rw-r--r-- 1 root root  832 9月   1 11:34 hrk.cert
-rw-r--r-- 1 root root 2916 9月   1 11:34 hsk_cek.cert
-rw-r--r-- 1 root root  832 9月   1 11:34 hsk.cert
-rw-r--r-- 1 root root  448 9月   1 11:45 report.bin
-rw-r--r-- 1 root root   32 9月   1 11:45 user_data.bin
```

(6) 解析度量报告，主要将报告中的版本、任务状态位映射、用户提供数据（与 user_data.bin 中数据相同）、度量任务哈希值、当前所有度量任务的状态信息（详细报告时才存在）进行解析显示，方便用户了解报告中的内容。

结果如下：

```
[root@localhost hygon]# hag tdm parse_tdm_report -report_file report.bin
############################TDM_REPORT START###################################
###version:                      0x10000
###fw_version:                   1882
###report_type:                  1
###task_nums:                    3
###task_bitmap:
0xe0 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

###task_error_bitmap:
0xe0 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

###task_running_bitmap:
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

###user_supplied_data_len: 32
###user_supplied_data:
0x63 0xd6 0xae 0x11 0xd9 0x20 0xff 0x64 0xac 0x7a 0xd9 0x30 0xe4 0x9d 0x0a 0xeb

0xad 0x84 0x0c 0x39 0x3a 0x8d 0x05 0x3c 0x0e 0x8d 0x08 0x76 0x30 0x5f 0x45 0xd3

###aggregate_hash:
```

```
0x0f 0x68 0xc1 0x01 0x3e 0xd9 0x4b 0x33 0x7c 0x77 0xf8 0x1f 0xde 0x9f 0x49 0xa3
0xff 0x76 0xad 0x40 0x4b 0xeb 0xe4 0x37 0xe4 0x80 0x35 0x5c 0x03 0x6e 0x34 0x60

**************************************************
###task_id:                  15
###perios_ms:                0
###measured_count:           30
###last_measure_elapsed_ms: 32071
###measured_hash:
0x69 0x20 0xde 0x37 0x20 0xc7 0x9e 0x44 0xba 0x82 0x7b 0xcb 0x4a 0x72 0xc7 0xbd
0xac 0xe6 0xd4 0xf2 0xe2 0xf5 0xd3 0x06 0x84 0x46 0x51 0x12 0x1f 0x8c 0xd7 0xde

**************************************************
###task_id:                  16
###perios_ms:                0
###measured_count:           24
###last_measure_elapsed_ms: 32000
###measured_hash:
0x7d 0x49 0x86 0x8a 0x49 0xc5 0xdd 0xd3 0xa3 0x3b 0x6b 0x42 0x16 0x75 0xc6 0x87
0x78 0xf9 0x16 0x36 0x33 0x91 0xc0 0x6e 0x47 0xbd 0x5f 0x55 0x21 0xba 0xcb 0xe9

**************************************************
###task_id:                  17
###perios_ms:                0
###measured_count:           29
###last_measure_elapsed_ms: 31946
###measured_hash:
0xe8 0x45 0xd4 0x8e 0xc1 0x3f 0x0c 0xc8 0xf5 0x22 0x8b 0xf5 0xe2 0x25 0x5e 0x8e
0xd7 0x9e 0xdd 0x04 0x72 0xcb 0xa4 0x0d 0x2b 0xa4 0xc4 0x4c 0x7d 0xe2 0xb5 0x3e

**************************************************

###sig_key_usage_id:          0x2001
###sig_r:
0x3b 0x4a 0xd3 0xac 0xae 0xb0 0x8e 0x55 0xa9 0xb4 0xf3 0x5e 0x66 0x86 0xf4 0x23
0xd2 0x5d 0x60 0x1a 0xee 0x02 0x88 0xea 0xc4 0x29 0xf0 0x17 0xcf 0x82 0x5c 0x01

###sig_s:
0xc4 0x96 0x5f 0xfb 0xe9 0xfc 0x76 0xd7 0x6d 0xea 0xf5 0x65 0x83 0x7c 0x8f 0x0c
0xa5 0xb1 0x1b 0x7e 0xae 0xde 0x9f 0x58 0xde 0xfd 0xb6 0xb0 0xd6 0x6f 0x13 0xc6

report origin data:
0x00 0x00 0x01 0x00 0x5a 0x07 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x03 0x00
0xe0 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

```
0xe0 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x00
0x63 0xd6 0xae 0x11 0xd9 0x20 0xff 0x64 0xac 0x7a 0xd9 0x30 0xe4 0x9d 0x0a 0xeb
0xad 0x84 0x0c 0x39 0x3a 0x8d 0x05 0x3c 0x0e 0x8d 0x08 0x76 0x30 0x5f 0x45 0xd3
0x0f 0x68 0xc1 0x01 0x3e 0xd9 0x4b 0x33 0x7c 0x77 0xf8 0x1f 0xde 0x9f 0x49 0xa3
0xff 0x76 0xad 0x40 0x4b 0xeb 0xe4 0x37 0xe4 0x80 0x35 0x5c 0x03 0x6e 0x34 0x60
0x0f 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x1e 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x47 0x7d 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x69 0x20 0xde 0x37 0x20 0xc7 0x9e 0x44 0xba 0x82 0x7b 0xcb 0x4a 0x72 0xc7 0xbd
0xac 0xe6 0xd4 0xf2 0xe2 0xf5 0xd3 0x06 0x84 0x46 0x51 0x12 0x1f 0x8c 0xd7 0xde
0x10 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x18 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x7d 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x7d 0x49 0x86 0x8a 0x49 0xc5 0xdd 0xd3 0xa3 0x3b 0x6b 0x42 0x16 0x75 0xc6 0x87
0x78 0xf9 0x16 0x36 0x33 0x91 0xc0 0x6e 0x47 0xbd 0x5f 0x55 0x21 0xba 0xcb 0xe9
0x11 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x1d 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0xca 0x7c 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0xe8 0x45 0xd4 0x8e 0xc1 0x3f 0x0c 0xc8 0xf5 0x22 0x8b 0xf5 0xe2 0x25 0x5e 0x8e
0xd7 0x9e 0xdd 0x04 0x72 0xcb 0xa4 0x0d 0x2b 0xa4 0xc4 0x4c 0x7d 0xe2 0xb5 0x3e
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x20 0x00 0x00
0x3b 0x4a 0xd3 0xac 0xae 0xb0 0x8e 0x55 0xa9 0xb4 0xf3 0x5e 0x66 0x86 0xf4 0x23
0xd2 0x5d 0x60 0x1a 0xee 0x02 0x88 0xea 0xc4 0x29 0xf0 0x17 0xcf 0x82 0x5c 0x01
0xc4 0x96 0x5f 0xfb 0xe9 0xfc 0x76 0xd7 0x6d 0xea 0xf5 0x65 0x83 0x7c 0x8f 0x0c
0xa5 0xb1 0x1b 0x7e 0xae 0xde 0x9f 0x58 0xde 0xfd 0xb6 0xb0 0xd6 0x6f 0x13 0xc6

##############################TDM_REPORT END###################################
parse tdm report successful.
parse_tdm_report command success!

[tdm] Command successful!
```

(7) 验证报告

结果如下：

```
1. [root@localhost hygon]# hag tdm verify_tdm_report -ak_cert ak.cert -
   report_file report.bin
2. verify tdm report successful.
3. verify_tdm_report command success!
4.
5. [tdm] Command successful!
```

(8) 可以看到成功使用 AK 证书验证了度量报告，由于 AK 证书已在上一个场景中经过验证，可以确定是可信的，这里使用可信的 AK 继续验证度量报告，验证通过后，可以确定报告也是可信的，用户可以根据此报告获取到 TDM 内的各种状态信息；该演示为其中的一种场景，用户也可以根据自己需求，来获取其他的各种场景下的度量报告进行验证。

(9) 卸载 tdm_verify 的模块，度量报告验证演示完成

```
(10)$ sudo rmmod tdm-verify.ko
```

TDM VPCR 获取与审计重放 VPCR 值验证流程（1.3 固件版本后支持）

该场景演示了获取 VPCR 值以及 VPCR 审计信息的原始 TDM 度量值与 fTPM 原始 PCR 值获取，以及根据审计信息重放 VPCR 值的验证流程。通过该场景，可以熟悉并验证 VPCR 机制的应用，理解 TDM 利用 fTPM 的 PCR 实现 VPCR 从而利用 fTPM 成熟的 PCR 远程证明的过程。hag 工具提供了 VPCR 审计信息获取的具体流程以及使用根据审计信息重放 VPCR 值的过程，用户可以通过使用 hag 来熟悉 TDM 基于 VPCR 的使用与审计信息接口的使用。

基本验证流程如下：

(1) TDM 支持 get_vPCR_audit、parse_vPCR_audit、replay_vPCR_audit 三个与 VPCR 报告相关的命令，其中 get_vPCR_audit 命令需要两个参数，第一个为放置 VPCR 审计信息的文件名字，这里以 audit.bin 为例，第二个为需要获取对应 PCR 的编号，这里以 PCR11 为例。parse_vPCR_audit 命令用来解析 get_vPCR_audit 命令获取的 audit.bin 文件，获取原始 TDM 度量任务度量值与 fTPM 的原始 PCR 值。replay_vPCR_audit 命令用于重放 VPCR，第一个参数为需要重放的 VPCR 文件，这里为 audit.bin，第二个参数可选，为重放生成文件，可以根据需求选择。

(2) 目前最新的 tdm_verify 经过调整后，可以支持 VPCR 的相关测试，目前 tdm_verify 共创建了三个度量任务，这里三个任务分别对应 PCR 编号的 PCR10，PCR11，PCR12，这里以获取 tdm_verify 的场景 1 任务更新操作流程中对应 PCR11 的审计信息为例进行说明。

(3) 首先通过 tpm2_tools 命令查看原始 PCR 的值（这里版本为 tpm2_pcrread 命令，其他版本可能为 tpm2_pcrlist，根据机器装的 tpm2_tools 的版本确定）

```
[hygon@localhost hygon]$ tpm2_pcrread sm3_256
sm3_256:
  0 : 0x5D25A693796A9D6060834A9FB0AF416E9C9FB4D47A22326BBC45686300B471A3
  1 : 0xFAC21DA05E1F8467972D6ABAF2CBED26FBB81B20A26A2751D32798EE574A7B1F
  2 : 0x0D72B0164E4FA67D6B43D3CB8EAD734737E479767E0D545EFF22C6FE6275B357
  3 : 0x0D72B0164E4FA67D6B43D3CB8EAD734737E479767E0D545EFF22C6FE6275B357
  4 : 0xA2381A4E30198BED49FB10A3D86274930B10D0EE788ED1121D1B83CF814362C9
  5 : 0xC71BED60766F8B89F6296C076F88E702B0E0474ACB8019AC8B8DB52E66E739ED
  6 : 0x0D72B0164E4FA67D6B43D3CB8EAD734737E479767E0D545EFF22C6FE6275B357
  7 : 0x2304AF3530A51BC03051BA7D3A2BB7B462120DF1B1D13BB55FA0B565831C19F4
  8 : 0xDEA0758622845043428A74802AABB23B53941CD216BA934FBFB5AF4D69FD7905
  9 : 0x2A12748335078EF34ABB0803C293AD390C2D9AB8B6D5FA9086E6A1ED4CD706FD
  10: 0x0000000000000000000000000000000000000000000000000000000000000000
  11: 0x0000000000000000000000000000000000000000000000000000000000000000
  12: 0x0000000000000000000000000000000000000000000000000000000000000000
  13: 0x0000000000000000000000000000000000000000000000000000000000000000
```

```
  14: 0x0000000000000000000000000000000000000000000000000000000000000000
  15: 0x0000000000000000000000000000000000000000000000000000000000000000
  16: 0x8E4D2AD793AEDBA2DBFE2AE09F0A49727988DFE8F460923A75B524A7FE9CFD80
  17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  23: 0x0000000000000000000000000000000000000000000000000000000000000000
```

(4) 可以看到原始 PCR11 的值为全 0，加载 tdm_verify 的场景 1 模块来创建带 VPCR 属性的 TDM 度量任务

```
$ sudo insmod tdm-verify.ko test_scene=1
```

(5) 重新查看 PCR 值。可以看到由于 tdm_verify 的三个任务分别对应 PCR10、PCR11、PCR12，分别拓展出了 VPCR 的 PCR10、PCR11、PCR12 值。

```
[hygon@localhost hygon]$ tpm2_pcrread sm3_256
sm3_256:
  0 : 0x5D25A693796A9D6060834A9FB0AF416E9C9FB4D47A22326BBC45686300B471A3
  1 : 0xFAC21DA05E1F8467972D6ABAF2CBED26FBB81B20A26A2751D32798EE574A7B1F
  2 : 0x0D72B0164E4FA67D6B43D3CB8EAD734737E479767E0D545EFF22C6FE6275B357
  3 : 0x0D72B0164E4FA67D6B43D3CB8EAD734737E479767E0D545EFF22C6FE6275B357
  4 : 0xA2381A4E30198BED49FB10A3D86274930B10D0EE788ED1121D1B83CF814362C9
  5 : 0xC71BED60766F8B89F6296C076F88E702B0E0474ACB8019AC8B8DB52E66E739ED
  6 : 0x0D72B0164E4FA67D6B43D3CB8EAD734737E479767E0D545EFF22C6FE6275B357
  7 : 0x2304AF3530A51BC03051BA7D3A2BB7B462120DF1B1D13BB55FA0B565831C19F4
  8 : 0xDEA0758622845043428A74802AABB23B53941CD216BA934FBFB5AF4D69FD7905
  9 : 0x2A12748335078EF34ABB0803C293AD390C2D9AB8B6D5FA9086E6A1ED4CD706FD
  10: 0x67B124D86989268B92A26FBB88CF440FD6157475BB5472A27ADEB0D77843FDB9
  11: 0xC4D7DDA8900CA0F784F36D9C8C95039244E842CB3FE73B5AFCCE521872D41D3F
  12: 0x2E2D1C8F7759D7CFB57004C007FC22EB3DB340AF7CAE90D5668E3AEA5A33F6D2
  13: 0x0000000000000000000000000000000000000000000000000000000000000000
  14: 0x0000000000000000000000000000000000000000000000000000000000000000
  15: 0x0000000000000000000000000000000000000000000000000000000000000000
  16: 0x8E4D2AD793AEDBA2DBFE2AE09F0A49727988DFE8F460923A75B524A7FE9CFD80
  17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  23: 0x0000000000000000000000000000000000000000000000000000000000000000
```

(6) 在 hag 工具的目录获取 PCR11 对应的 TDM 度量任务度量值与 fTPM 的原始 PCR 值，可以看到成功获取 bin 的审计文件。

```
[root@localhost hygon]# hag tdm get_vPCR_audit -audit_file audit.bin -pcr_num 11
######### PCR number   : 11 #########
get vPCR audit successful.
get_vPCR_audit command success!

[tdm] Command successful!
```

(7) 解析审计信息，主要获取原始 TPM 的 PCR 值与 PCR 对应 TDM 度量任务的信息。可以看到 TPM 原始信息与先前获取到的原始信息一致。

```
[root@localhost hygon]# hag tdm parse_vPCR_audit -audit_file audit.bin
############################TDM_VPCR_AUDIT START#################################
###pcr:                      11
###tpm2_digest:
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

**************************************************
###task_id:                  19
###hash:
0x7d 0x49 0x86 0x8a 0x49 0xc5 0xdd 0xd3 0xa3 0x3b 0x6b 0x42 0x16 0x75 0xc6 0x87

0x78 0xf9 0x16 0x36 0x33 0x91 0xc0 0x6e 0x47 0xbd 0x5f 0x55 0x21 0xba 0xcb 0xe9

**************************************************
############################TDM_VPCR AUDIT END#################################

parse vPCR audit successful.
parse_vPCR_audit command success!
```

(8) 重放 VPCR，如下，重放出的 VPCR 值与 PCR11 的值一致，重放成功。

```
[root@localhost hygon]# hag tdm replay_vPCR_audit -vPCR_file audit.bin
############################TDM_VPCR_AUDIT_REPLAY START#########################
########
###replay pcr:               11
VPCR hash:
0xc4 0xd7 0xdd 0xa8 0x90 0x0c 0xa0 0xf7 0x84 0xf3 0x6d 0x9c 0x8c 0x95 0x03 0x92

0x44 0xe8 0x42 0xcb 0x3f 0xe7 0x3b 0x5a 0xfc 0xce 0x52 0x18 0x72 0xd4 0x1d 0x3f
```

```
7.
8.  ############################TDM_VPCR_AUDIT_REPLAY END#############################
    ######
9.  replay vPCR audit successful.
10. replay_vPCR_audit command success!
11.
12. [tdm] Command successful!
```

(9) VPCR 审计主要用来获取度量的详细信息且确保审计信息不被篡改，至此，VPCR 属性的度量基本验证流程结束。

(10) 卸载 tdm_verify 的模块

```
1.  $ sudo rmmod tdm-verify.ko
```

4.2 Keylime 最佳实践

4.2 .1 keylime 概述

Keylime 是一个利用可信计算 TPM 技术的开源可扩展信任系统。Keylime 已经进入 CNCF 项目且被 Redhat 等多个主流发行版集成。Keylime 提供了一种端到端解决方案，用于为远程计算机引导基于硬件的加密信任、加密负载的配置以及运行时系统完整性监控。 它还为任何给定 PCR（平台配置寄存器）的远程证明提供了灵活的框架。 用户可以创建自己的自定义操作，当机器未通过其验证度量时将触发这些操作。

Keylime 的使命是让开发人员和用户能够轻松使用 TPM 技术，而无需深入了解 TPM 较低级别的操作。 在许多场景中，租户需要对于不受自己完全控制的机器的远程证明（例如混合云的消费者或位于不安全的、容易被篡改的物理位置的远程边缘/物联网设备）。

通过 CLI 应用程序和一组 RESTful APIs(包括 http 和 https，其中 https 相关的 RESTful APIs 采用 mTLS 握手协议) 来执行和管理 keylime。

Keylime 由三个主要组件组成；verifier、registrar 和 agent。

- verifer 持续验证运行 agent 的计算机的完整性状态。
- registrar 是在 Keylime 中注册的所有 agent 的数据库，并托管 TPM 供应商的公钥。
- agent 部署在要监控的 TPM 机器上。

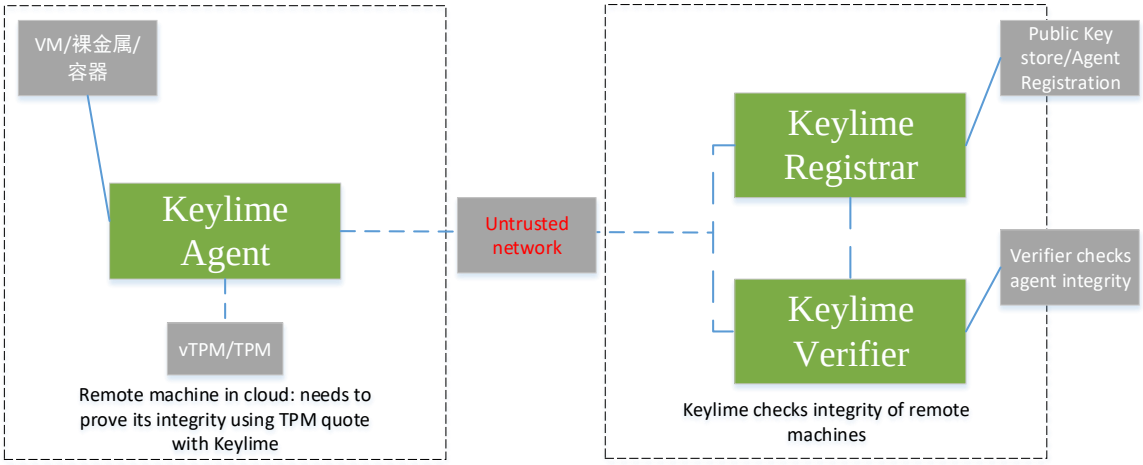


图 4-2-1 keylime 架构图

此外 keylime 还提供 tenant 工具便于用户远程管理 agent。

4.2.2 龙蜥社区在 keylime 社区的工作与探索

龙蜥社区自其可信计算 SIG 成立以来，一直在关注可信计算业界进展和国际 OSV 厂商的可信计算方案。同时龙蜥社区在 keylime 社区积极贡献代码与适配，一共在 rust-keylime 和 keylime 两个仓库提交并合入 17 个 patch，包括多个

features、bugfixes 和文档。详见 keylime release notes 和 rust-keylime, 具体包括：

- features：集成龙蜥 anolis 以及下游阿里云 Alibaba Cloud Linux OS 的安装代码、集成阿里云 vTPM EK 证书、支持 keylime 安装时选择缺省的监听端口等。
- bugfixes: 修复 measure boot 时处理部分 eventlog 出错、修复 rust-keylime 跟 keylime RESTful APIs 版本和接口不一致、移除无用的代码等。
- 文档：修复安装文档和实践文档中多处命令错误等。

表 4-2-1 龙蜥社区在 keylime 社区的贡献

开源软件名称	总计 commit 数量	总计修改行数
rust-keylime	3	-19/+20
keylime	14	-24/+156

在完成 keylime 的适配和实践后，龙蜥社区也将自己的 keylime 经验写入到白皮书中。未来，龙蜥社区除了继续加强与 keylime 社区的沟通和贡献（参与 keylime rust 化）外，还将结合自己在国密、机密计算的积累围绕 keylime 开展一些国密、、机密计算相关的工作，尽情期待。

4.2.3 龙蜥 Anolis OS 上 keylime 用途与实践

Keylime 可以借助 PCR 或者 Measure boot 监控远程机器（Agent 部署机器）的启动时的状态（完整性等）和借助 IMA 来监控运行时的完整性。开启对应的策略（policy）后，时刻轮询监控着对应 agent 的状态，如果发现异常则返回给 verifier

执行对应的操作（标记失败/停止轮询/打印错误等）。关于这部分的用法详见下文 Anolis OS 上 keylime 高级功能实践章节。

Keylime 也可以通过 RESTful APIs 去监控/管理/查询 Keylime 的 agent、verifier 以及 registrar，便于用户以及运维人员有效的管理 Keylime 的各个组件以及验证远程机器的完整性等。此外keylime提供更加安全的mTLS协议和基于Https的 RESTful APIs。其中如果被正确执行，则 RESTful APIs 返回对应的信息且状态码为 200，否则则为错误运行。关于这些 APIs 的用法详见下文用 Restful API 去监控/管理 Anolis OS 上的各个 keylime 组件章节。

Anolis OS 上 keylime 安装与配置、运行

安装（以 anolis 8.8 为例）

keylime 分为两个代码仓库：

- keylime：包含除了 agent 以外的其它 keylime 组件（verifier，registrar，tenant），下载后执行 cd keylime && ./installer.sh -i 命令进行安装；
- rust-keylime： 包含 keylime 的 agent 组件；

rust-keylime 安装步骤如下：

1）安装 tpm2-tss 软件包（**如果 keylime agent 跟 keylime 其它组件安装在一台机器，则这一步可以省略**）；

2）安装 tpm2-tools 软件包（如果 keylime agent 跟 keylime 其它组件安装在一台机器，则这一步可以省略）。

3）**安装 rust-keylime：**可以根据以下命令在 anolis（以 anolis 8.8 为例）上安装 rust-keylime（keylime agent）。

```
yum install -y libarchive-devel clang-devel rust cargo openssl-devel jq
git clone https://github.com/keylime/rust-keylime.git
cd rust-keylime
cargo build
make install
useradd keylime
mkdir -p /var/lib/keylime/cv_ca
# 将 keylime verifier 机器上的/var/lib/keylime/cv_ca/cacert.crt 拷贝到 agent
# 机器上/var/lib/keylime/cv_ca/目录下，以便于后续 Agent 侧 https RESTful APIs 的访问
chown -R keylime /var/lib/keylime
```

配置

- verifier 配置:/etc/keylime/verifier.conf 为 verifier 的缺省配置。一般情况下不需要进行修改（可根据实际需求修改）。
- registrar 配置:/etc/keylime/registrar.conf 为 registrar 的缺省配置。一般情况下不需要进行修改（可根据实际需求修改）。
- agent 配置:/etc/keylime/agent.conf 为 agent 的缺省配置。当 agent 跟 verifier、registrar 部署在同一台机器时，不需要修改 agent 的配置；否则需要修改 agent 监听的 IP、contact_ip(verifier 和 tenant 用来连接的 agent IP)、registrar 的 IP 以便于正确注册和通信。
- tenant 配置:/etc/keylime/tenant.conf 为 tenant 的缺省配置。一般情况下不需要进行修改（可根据实际需求修改）。

运行

启动方式：

1）以二进制方式启动 verifier、registrar 和 agen；

2）以 systemd 方式启动 verifier、registrar（具体命令如下）。

```
cd keylime
./services/installer.sh
systemctl start keylime_verifier
systemctl start keylime_registrar
systemctl start keylime_agent
```

Anolis OS 上 keylime 高级功能实践

使用用户选择的 PCR 进行监控

该功能需要 Agent 侧有 TPM，但因为 TPM 的 PCR 数量有限，扩展性不好。配置 tpm_policy 并用 keylime_tenant 工具进行添加，具体命令如下：

```
keylime_tenant -v 121.43.60.253 -t 120.26.100.138 \
    --uuid d432fbb3-d2f1-4a97-9ef7-75bd81c00000 \
    --tpm_policy "{\"15\": [\"0000000000000000000000000000000000000000\", \
     \"0000000000000000000000000000000000000000000000000000000000000000\", \
\"0000000000000000000000000000000000000000000000000000000000000000000000000000\
00000000000000000000\"]}" \
    -c add --cert /var/lib/keylime/cv_ca
```

监控

成功的 case 如下（agent 时刻监控 TPM PCRs 的状态）：

```

DEBUG keylime_agent::quotes_handler  > Calling Integrity Quote with nonce: lCYwD
xi2UkTIWM6Fk2aH, mask: 0x408000
 INFO  keylime_agent::quotes_handler  > GET integrity quote returning 200 respon
se
 INFO  actix_web::middleware::logger  > GET
                /v2.1/quotes/integrity?nonce=lCYwDxi2UkTIWM6Fk2aH&mask=0x408000&
partial=1&ima_ml_entry=0
                HTTP/1.1 from 121.43.60.253 result 200 (took 1229.894715 ms)
 INFO  keylime_agent                 > GET invoked from "121.43.60.253" with ur
i
                /v2.1/quotes/integrity?nonce=WSn8mEpGLjN5I8mhHjPn&mask=0x408000&
partial=1&ima_ml_entry=0
```

以下为监控 Measured Boot。

监控 MeasureBoot 功能：

- 需要 Agent 有 TPM 以及使能 IMA；
- Keylime 提供脚本 create_mb_refstate 生成对应的 measured boot reference state 和 policy(根据/sys/kernel/security/tpm0/binary_bios_measurements 里面的 boot event log)；
- Keylime 提供 keylime_tenant（-mb_refstate 参数）添加对应的参考值和 policy，然后启动 Agent。

如下图所示，监控功能主要包括三个步骤：

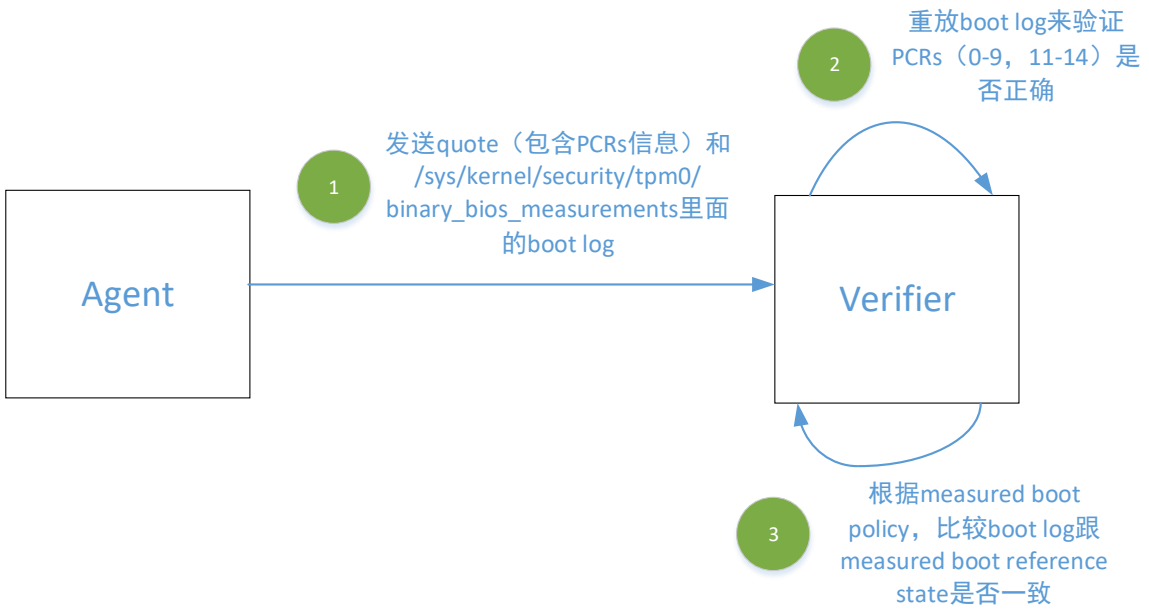


图 4–2-2 keylime measure boot 流程图

（1）生成 measure boot policy：

```
cd keylime
./scripts/create_mb_refstate \
-i /sys/kernel/security/tpm0/binary_bios_measurements \
    ./measured_boot_reference_state.json
```

```
cat ./measured_boot_reference_state.json | jq .
```

（2）用 keylime_tenant 工具进行添加 mb_reference，对应的命令如下：

```
keylime_tenant -c update -t 120.26.100.138 -v 121.43.60.253 \
    -u d432fbb3-d2f1-4a97-9ef7-75bd81c00000 \
    --mb_refstate ./measured_boot_reference_state.json \
    --cert /var/lib/keylime/cv_ca
```

（3）agent 侧查看轮询结果：时刻轮询/监控着是否有异常。

```
INFO  keylime_agent                 > GET invoked from "121.43.60.253" with uri
                 /v2.1/quotes/integrity?nonce=EiRTFv9tgNBmwy6HwxAC&mask=0xfbff&partial=1&ima_ml_entry=0
DEBUG keylime_agent::quotes_handler  > Calling Integrity Quote with nonce: EiRTFv9tgNBmwy6HwxAC, mask: 0xfbff
INFO  keylime_agent::quotes_handler  > GET integrity quote returning 200 response
INFO  actix_web::middleware::logger  > GET
               /v2.1/quotes/integrity?nonce=EiRTFv9tgNBmwy6HwxAC&mask=0xfbff&partial=1&ima_ml_entry=0
               HTTP/1.1 from 121.43.60.253 result 200 (took 1452.270707 ms)
```

Runtime Integrity Monitoring

该功能：

- 需要 Agent 有 TPM 以及使能 IMA
- Keylime 提供脚本 keylime_create_policy/create_runtime_policy.sh 创建对应的 runtime policy
- Keylime 提供 keylime_tenant（-runtime-policy 参数）添加对应的 runtime policy，然后启动 Agent

使用 keylime_create_policy 工具来生成 policy：

```
keylime_create_policy -m /sys/kernel/security/ima/ascii_runtime_measurements -o runtime_policy.json
cat runtime_policy.json | jq .
```

使用 keylime_tenant 来添加 runtime_policy:

```
keylime_tenant -c update --uuid d432fbb3-d2f1-4a97-9ef7-75bd81c00000 \
       -t 120.26.100.138 -v 121.43.60.253 --runtime-
policy /root/runtime_policy.json \
       --runtime-policy-name=tpm --cert /var/lib/keylime/cv_ca
```

查看 runtime policy

```
## curl --key /var/lib/keylime/cv_ca/client-private.pem \
      --cert /var/lib/keylime/cv_ca/client-cert.crt \
      -k "https://127.0.0.1:8881/v2.1/allowlists/tpm"  | jq .
{
  "code": 200,
  "status": "Success",
  "results": {
    "name": "tpm",
    "tpm_policy": null,
    "runtime_policy": ...
}
```

监控 IMA 错误

从 verifier 的日志可以看到有一些没有进行 IMA 签名，无法验证，所有直接报错

```
2023-09-
07 15:20:10.295 - keylime.tpm - INFO - Checking IMA measurement list on agent:

   d432fbb3-d2f1-4a97-9ef7-75bd81c00000
2023-09-
07 15:20:10.295 - keylime.ima - WARNING - Hashes for file boot_aggregate don't m
atch
  1aa841ace294d93414158a2f070c92d078c464da0110269d3ad1e59367cdc285 not in
  ['fd2cf72bae331c6ba3db242e04f65ad5ca5c9da3f94dd5c78f5e56496e7cf0da']
2023-09-
07 15:20:10.296 - keylime.ima - ERROR - IMA ERRORS: Some entries couldn't be val
idated. Number of
  failures in modes: ImaSig 1.
2023-09-07 15:20:10.357 - keylime.verifier - WARNING - Agent d432fbb3-d2f1-4a97-
9ef7-75bd81c00000 failed,
  stopping polling
```

用 Restful API 去监控/管理 Anolis OS 上的各个 keylime 组件

registrar

使用 registrar 的 RESTful APIs 能够对 agent 进行注册、查询、删除、激活等操作。注意以下示例中的 registrar IP 需要根据实际 IP 进行修改。

GET /v2.1/agents/

用来获取注册的 agents 列表，具体命令如下：

```
# curl --key /var/lib/keylime/cv_ca/client-private.pem \
      --cert /var/lib/keylime/cv_ca/client-cert.crt \
      -k "https://127.0.0.1:8891/v2.1/agents"  | jq .
{
  "code": 200,
  "status": "Success",
  "results": {
    "uuids": [
      "d432fbb3-d2f1-4a97-9ef7-75bd81c00000"
    ]
  }
}
```

GET /v2.1/agents/{agent_id:UUID}

获取对应 agent 的端口、IP、EK 证书等信息, 命令如下：

```
# curl --key /var/lib/keylime/cv_ca/client-private.pem \
      --cert /var/lib/keylime/cv_ca/client-cert.crt \
      -k "https://127.0.0.1:8891/v2.1/agents/d432fbb3-d2f1-4a97-9ef7-
75bd81c00000"  | jq .
{
  "code": 200,
  "status": "Success",
  "results": {
    ...
    "ip": "121.43.60.253",
    "port": 9002,
    "regcount": 1
  }
}
```

PUT /v2.1/agents/{agent_id:UUID}/activate

激活 agent_id 的 agent，注意这是一个 http 请求，不是 https，如果用 https 会提示这个不是 TLS 接口。命令如下:

```
# curl -k \
-X PUT "http://127.0.0.1:8890/v2.1/agents/d432fbb3-d2f1-4a97-9ef7-75bd81c00000/activate" \
       -H 'Content-Type: application/json' \
   -d '{"auth_tag":"166be150040c57b4e2c69ad7a5dd4c57059e5838a1df4715872f6e385e8ce1ed91"}' \
      | jq .
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

DELETE /v2.1/agents/{agent_id:UUID}

从 registrar 中移除 ID 为 agent_id 的 agent，移除后再查看发现没有该 agent 了, 使用该命令

```
# curl -k \
-X PUT "http://127.0.0.1:8890/v2.1/agents/d432fbb3-d2f1-4a97-9ef7-75bd81c00000/activate" \
       -H 'Content-Type: application/json' \
   -d '{"auth_tag":"166be150040c57b4e2c69ad7a5dd4c57059e5838a1df4715872f6e385e8ce1ed91"}' \
      | jq .
{
  "code": 200,
  "status": "Success",
  "results": {}
}
# curl --key /var/lib/keylime/cv_ca/client-private.pem \
     --cert /var/lib/keylime/cv_ca/client-cert.crt \
     -k "https://127.0.0.1:8891/v2.1/agents"  | jq .
{
  "code": 200,
  "status": "Success",
  "results": {
    "uuids": []
  }
}
```

POST /v2.1/agents/{agent_id:UUID}

注册 agent_id 的 agent 到 registrar.**这是一个 http 不是 https 的请求。** 注册及注册后的查询命令如下

```
# curl -X POST  "http://127.0.0.1:8890/v2.1/agents/d432fbb3-d2f1-4a97-9ef7-75bd81c00000"
  -H 'Content-Type: application/json' \
  -d '{ "ekcert": "MIIE3DCCA8SgAwIBAgIBBDANBgkqhkiG9w0BAQsFADBvMQswCQYDVQQGEwJDTjEPMA0GA1UECgwGQWxpeXVuMTIwMA" \
   "aik_tpm": "ARgAAQALAAUAcgAAABAAFAALCAAAAAAAAQC7R7SiAAExqqCZJ60cTJXxcYMCRsctsh96vX/f2T31DMrB6SnCMV9euHlMUCUs" \
"ip": "127.0.0.1","port": 9002}' | jq .
{
  "code": 200,
  "status": "Success",
  ...
}
# curl --key /var/lib/keylime/cv_ca/client-private.pem --cert /var/lib/keylime/cv_ca/client-cert.crt \
   -k "https://127.0.0.1:8891/v2.1/agents"  | jq .
{
  "code": 200,
  "status": "Success",
  "results": {
    "uuids": [
      "d432fbb3-d2f1-4a97-9ef7-75bd81c00000"
    ]
  }
}
```

verifier

GET /v2.1/agents/{agent_id:UUID}

从 CV 中获取 agent agent_id 的状态。具体命令如下：

```
# curl --key /var/lib/keylime/cv_ca/client-private.pem \
      --cert /var/lib/keylime/cv_ca/client-cert.crt \
      -k "https://127.0.0.1:8881/v2.1/agents/d432fbb3-d2f1-4a97-9ef7-75bd81c00000"  | jq .
{
  "code": 200,
  "status": "Success",
  "results": {
    ...
    "hash_alg": "sha256",
    "enc_alg": "rsa",
    "sign_alg": "rsassa",
    "verifier_id": "default",
```

```
    "verifier_ip": "121.43.60.253",
    "verifier_port": 8881,
    "severity_level": 6,
  }
}
```

PUT /v2.1/agents/{agent_id:UUID}/stop

停止对 agent_id 的 cv 轮询，但不要删除（对于已经启动的 agent_id）。具体命令如下：

```
# curl --key /var/lib/keylime/cv_ca/client-private.pem \
      --cert /var/lib/keylime/cv_ca/client-cert.crt -k \
    -X PUT "https://127.0.0.1:8881/v2.1/agents/d432fbb3-d2f1-4a97-9ef7-75bd81c00000/stop" \
     | jq .
{
  "code": 200,
  "status": "Success",
  "results": {}
}
```

DELETE /v2.1/agents/{agent_id:UUID}

删除 agent_id 实例。删除包括删除后的查看命令如下：

```
# curl --key /var/lib/keylime/cv_ca/client-private.pem \
      --cert /var/lib/keylime/cv_ca/client-cert.crt \
   -k -X DELETE "https://127.0.0.1:8881/v2.1/agents/d432fbb3-d2f1-4a97-9ef7-75bd81c00000" \
      | jq .
{
  "code": 200,
  "status": "Success",
  "results": {}
}
# curl --key /var/lib/keylime/cv_ca/client-private.pem \
      --cert /var/lib/keylime/cv_ca/client-cert.crt \
      -k "https://121.0.0.1:8881/v2.1/agents/d432fbb3-d2f1-4a97-9ef7-75bd81c00000"  | jq .
{
  "code": 404,
  "status": "agent id not found",
  "results": {}
}
```

GET /v2.1/allowlists/{runtime_policy_name:string}

从 CV 中检索命名的运行时策略 runtime_policy_name。比如 tpm 的 policy 创建了，可以通过以下命令查看

```
curl --key /var/lib/keylime/cv_ca/client-private.pem \
     --cert /var/lib/keylime/cv_ca/client-cert.crt -k \
     "https://127.0.0.1:8881/v2.1/allowlists/tpm"  | jq .
{
  "code": 200,
  "status": "Success",
  "results": {
    "name": "tpm",
    "tpm_policy": null,
    "runtime_policy": ...
  }
}
```

对于没有创建的 test policy，查询是没有的

```
# curl --key /var/lib/keylime/cv_ca/client-private.pem \
     --cert /var/lib/keylime/cv_ca/client-cert.crt -k \
     "https://127.0.0.1:8881/v2.1/allowlists/test"  | jq .
{
  "code": 404,
  "status": "Runtime policy test not found",
  "results": {}
}
```

DELETE /v2.1/allowlist/{runtime_policy_name:string}

删除 IMA policy runtime_policy_name. 比如删除已有的 tpm policy，然后再测试，发现该 policy 没了

```
# curl --key /var/lib/keylime/cv_ca/client-private.pem \
     --cert /var/lib/keylime/cv_ca/client-cert.crt \
     -k "https://127.0.0.1:8881/v2.1/allowlists/tpm"  | jq .
{
  "code": 200,
  "status": "Success",
  "results": {
    "name": "tpm",
    "tpm_policy": null,
    "runtime_policy": ...
  }
}
# curl --key /var/lib/keylime/cv_ca/client-private.pem \
     --cert /var/lib/keylime/cv_ca/client-cert.crt -X DELETE \
```

```
15.      -k "https://127.0.0.1:8881/v2.1/allowlists/tpm"  | jq .
16. # curl --key /var/lib/keylime/cv_ca/client-private.pem \
17.      --cert /var/lib/keylime/cv_ca/client-cert.crt \
18.      -k "https://127.0.0.1:8881/v2.1/allowlists/test"  | jq .
19. {
20.   "code": 404,
21.   "status": "Runtime policy test not found",
22.   "results": {}
23. }
```

而删除一个不存在的 policy test，会报错

```
1.  # curl --key /var/lib/keylime/cv_ca/client-private.pem \
2.       --cert /var/lib/keylime/cv_ca/client-cert.crt -X DELETE \
3.       -k "https://127.0.0.1:8881/v2.1/allowlists/test"  | jq .
4.  {
5.    "code": 404,
6.    "status": "Runtime policy test not found",
7.    "results": {}
8.  }
```

agent

GET /version

获取 agent 支持的 API 版本。对应的命令如下:

```
1.  # curl --key /var/lib/keylime/cv_ca/client-private.pem \
2.       --cert /var/lib/keylime/cv_ca/client-cert.crt -k \
3.       "https://127.0.0.1:9002/version" | jq .
4.  {
5.    "code": 200,
6.    "status": "Success",
7.    "results": {
8.      "supported_version": "2.1"
9.    }
10. }
```

GET /v2.1/keys/pubkey

获取 agent 的公钥. 对应的命令如下:

```
1.  # curl --key /var/lib/keylime/cv_ca/client-private.pem \
2.       --cert /var/lib/keylime/cv_ca/client-cert.crt -k \
3.       "https://127.0.0.1:9002/v2.1/keys/pubkey"  | jq .
4.  {
5.    "code": 200,
6.    "status": "Success",
7.    "results": {
8.      "pubkey": ...
```

```
9.    }
10. }
```

GET /v2.1/quotes/identity

从节点获取 identity quote, 对应的命令如下:

```
1.  # curl --key /var/lib/keylime/cv_ca/client-private.pem \
2.       --cert /var/lib/keylime/cv_ca/client-cert.crt -k \
3.       "https://127.0.0.1:9002/v2.1/quotes/identity?nonce=1234567890ABCDEFHIJ" \
4.       | jq .
5.  {
6.    "code": 200,
7.    "status": "Success",
8.    "results": {
9.      "quote": ...
10.     "hash_alg": "sha256",
11.     "enc_alg": "rsa",
12.     "sign_alg": "rsassa",
13.     "pubkey": ...
14.   }
15. }
```

GET /v2.1/quotes/integrity

从节点获取 integrity quote，具体命令如下：

```
1.  # curl --key /var/lib/keylime/cv_ca/client-private.pem  \
2.       --cert /var/lib/keylime/cv_ca/client-cert.crt -k \
3.  "https://127.0.0.1:9002/v2.1/quotes/integrity?nonce=1234567890&mask=0x10401&part
    ial=0"
4.  {
5.    "code": 200,
6.    "status": "Success",
7.    "results": {
8.      "quote": ...
9.      "hash_alg": "sha256",
10.     "enc_alg": "rsa",
11.     "sign_alg": "rsassa",
12.     "pubkey": ...
13.     "ima_measurement_list": ...
14.     "mb_measurement_list": ...
15.     "ima_measurement_list_entry": 0
16.   }
17. }
```

GET /v2.1/keys/verify

获取 bootstrap key 的验证, 对应的命令如下:

```
# curl --key /var/lib/keylime/cv_ca/client-private.pem \
      --cert /var/lib/keylime/cv_ca/client-cert.crt -k \
     "https://127.0.0.1:9002/v2.1/keys/verify?challenge=1234567890ABCDEFHIJ"  | jq .
{
  "code": 400,
  "status": "Bootstrap key not yet available.",
  "results": {}
}
```

4.3 基于可信根的全盘加密

本方案的撰写与实践参考了 Code Sample: Protecting secret data and keys using Intel® Platform... [7]

LUKS（Linux Unified Key Setup）是 Linux 硬盘加密的标准。LUKS 是由 Clemens Fruhwirth 在 2004 年创建的磁盘加密规范，最初用于 Linux，它是一种知名的、安全的、高性能的磁盘加密方法，基于改进版本的 cryptsetup，使用 dm-crypt 作为磁盘加密后端。LUKS 提供多种加密算法、多种加密模式和多种哈希函数可供选择，有 40 多种可能的组合。 磁盘加密可以防止存储设备安装在由攻击者控制的备用操作环境中，攻击者可以观察或篡改敏感信息。

本节以一个未受保护的 loopback-mounted 文件系统为例，介绍如何通过 TPM 增强基于 luks 磁盘加密的安全防护能力。

4.3.1 未受保护的文件系统

为了便于理解、本文假设该 loopback-mounted 文件系统是操作系统在引导时自动挂载的逻辑磁盘分区。

```
# 创建磁盘镜像，写入内容
# 如下命令，我们在磁盘中添加了名为 plain.txt 的文件，该文件包含的内容为“this is my plain text”
dd if=/dev/zero of=plain.disk bs=1M count=10
mkfs.ext4 plain.disk
mkdir mountpoint
sudo mount plain.disk mountpoint
```

```
sudo sh -c 'echo "This is my plain text" > mountpoint/plain.txt'
sudo umount mountpoint

# 可以通过简单的 linux 命令(如下所示）即可查看文件内容
strings plain.disk
```

上述命令会泄露文件名和敏感文件内容。后面的小节将添加保护，以防止攻击者执行上面的简单攻击。

4.3.2 基于 luks 的磁盘数据加密

龙蜥操作系统支持基于 luks 加密的磁盘卷。LUKS 可以保护静态数据的机密性。LUKS 中配置信息一般存储在分区表头中，以方便迁移。LUKS 卷可以自动挂载，默认的情况下加密密码口令可以通过人机交互的方式提供，或者通过指定为密钥文件(命令行参数)提供。cryptsetup 用户态程序可用于创建和管理 LUKS 卷。

```
# 创建一个新的 luks 卷，使用简单的密码口令作为该卷的保护密钥
dd if=/dev/zero of=enc.disk bs=1M count=50
dd if=/dev/urandom of=disk.key bs=1 count=32
sudo losetup /dev/loop0 enc.disk
sudo cryptsetup --key-file=disk.key luksFormat /dev/loop0

# 基于上述方案加密后，磁盘中的文件不再可见、起到了一定的保护作用，采用如下命令无法查看明文信息
strings enc.disk | grep -i plain
```

基于 luks 加密磁盘卷的方法的最大问题是，如果密码口令或密钥文件以纯文本形式存储在磁盘上，则 LUKS 保护很容易被破坏。因此，许多 LUKS 实现以交互方式提示输入密码口令。但是在许多物联网(IoT)和嵌入式等场景中输入口令的方式不可行。人工输入的一种有效替代方法是将密码口令密封在锚定平台状态的安全令牌上，如 TPM。

4.3.3 基于 TPM 安全存储空间保护 LUKS 的密码口令

由于自动挂载需要在运行时在没有用户干预的情况下向 cryptsetup 提供密码口令或密钥，因此它必须在某种程度上清晰可用。需要一个方法来加密密码口令或密钥，以便在需要它时来解密卷。这种方法还必须防止磁盘与安装它的系统分离的情况。这种机制就是 TPM。 该方案分为两个步骤:

- 将密码口令或密钥文件密封到 TPM 中。
- 在内存中解密秘密并将其传递给 cryptsetup。

```
# 创建并保存一个密封对象，并使用它来密封一个随机字节序列作为磁盘密钥：
tpm2_createprimary -Q --hierarchy=o --key-context=prim.ctx
dd if=/dev/urandom bs=1 count=32 status=none | tpm2_create --hash-algorithm=sha256 --public=seal.pub --private=seal.priv --sealing-input=- --parent-context=prim.ctx
tpm2_load -Q --parent-context=prim.ctx --public=seal.pub --private=seal.priv --name=seal.name --key-context=seal.ctx
tpm2_evictcontrol --hierarchy=o --object-context=seal.ctx 0x81010002

# 安装新密钥以取代旧密钥，并删除之前创建的旧密钥：
tpm2_unseal -Q --object-context=0x81010002 | sudo cryptsetup --key-file=disk.key luksChangeKey enc.disk
shred disk.key
rm -f disk.key

# 用 TPM 中密封的新身份验证挂载卷：
sudo losetup /dev/loop0 enc.disk
tpm2_unseal -Q --object-context=0x81010002 | sudo cryptsetup --key-file=- luksOpen /dev/loop0 enc_volume
sudo mount /dev/mapper/enc_volume mountpoint

# 磁盘访问被授予新的秘密：
ls mountpoint

# 卸载磁盘：

sudo umount mountpoint
sudo cryptsetup remove enc_volume
sudo losetup -d /dev/loop0
```

该方案将防止通过克隆或物理窃取加密磁盘进行的离线攻击。攻击者现在还需要平台上的 TPM 和磁盘来访问数据，因为解密密钥不在磁盘上——它安全地存储在

特定平台上的TPM 上。但是，攻击者仍然有可能从其他引导介质引导系统，执行上述命令，并检索解密 LUKS 卷所需的秘密。所需要的是一种可以锚定到受信任系统状态的身份验证机制。这可以通过 TPM 及其平台配置寄存器(PCR)密封提供来实现。

4.3.4 基于 PCR 增强授权保护带有密码口令的 LUKS

TPM 平台配置寄存器(PCR)用于防止攻击者更改系统引导参数或引导到其选择的操作系统以访问敏感数据。有效地使用 TPM PCR 需要系统固件、引导加载程序、操作系统内核和应用程序的配合。如果没有固件、驱动程序和软件来配置 TPM, TPM 将只是在 I/O 总线上处于非活动状态，什么也不做。 TPM PCR 用于使用安全散列算法(如 SHA-256)度量启动组件。当系统重置时，TPM PCR 默认为零值。当系统启动时，关键系统组件(如固件、BIOS、OS 加载程序等)的度量在启动过程中扩展到 PCR 中。扩展 PCR 是一个仅追加操作，并且需要对 TPM 进行 I/O 操作。因为不可能将 PCR 设置为用户指定的值，也不可能“收回” I/O，所以 TPM PCR 可以验证系统启动顺序，从而验证平台的状态，直到 PCR 度量停止为止。例如，PCR0 包含系统固件和 BIOS 的度量，但不包含操作系统引导加载程序或内核。使用 PCR0 且仅使用 PCR0 的后果是，本示例只能防止固件替换攻击，前提是原始固件正确地进行了度量，并且替换固件不会将度量结果伪造给 tpm(需要单独缓解的攻击)。

在实践中，有效的 PCR 集必须足够完整，以验证当前运行的代码和设计人员认为安全关键的预先运行的任何代码。《TCG PC Client Specific Implementation Specification for Conventional BIOS》仅指定了 PCR 0-7、16 和 23 的用法。操作系统引导期间和之后的 PCR 使用是特定于操作系统自定义的。

该方案有在上一方案的基础上增加了两个额外的步骤:

- 使用 PCR 作为 TPM 密封对象的身份验证授权代理（proxy authentication）。
- 在解密密钥文件后扩展密封 PCR，使其不能在当前引导中再次解密。

```
# 在 sha256 bank 中创建一个当前值为 PCR0 的 PCR 策略:
tpm2_startauthsession --session=session.ctx
tpm2_policypcr -Q --session=session.ctx --pcr-list="sha256:0" --
policy=pcr0.sha256.policy
tpm2_flushcontext session.ctx

# 现在将 TPM 非易失性内存中保护磁盘加密密钥的密封对象替换为一个新对象，该对象添加了我们
刚刚创建的 pcr 策略，作为访问密封密钥的身份验证机制:
tpm2_unseal --object-context=0x81010002 | tpm2_create -Q --hash-
algorithm=sha256 --public=pcr_seal_key.pub --private=pcr_seal_key.priv --
sealing-input=- --parent-context=prim.ctx --policy=pcr0.sha256.policy
tpm2_evictcontrol --hierarchy=o --object-context=0x81010002
tpm2_load -Q --parent-context=prim.ctx --public=pcr_seal_key.pub --
private=pcr_seal_key.priv --name=pcr_seal_key.name --key-
context=pcr_seal_key.ctx
tpm2_evictcontrol --hierarchy=o --object-
context=pcr_seal_key.ctx 0x81010002

# 现在尝试再次挂载加密磁盘，只不过这次密钥被密封在 TPM 对象中，其解密封操作只能通过满足
PCR 策略来访问。换句话说，通过 PCR 值所反映的预期系统软件状态不变来进行身份验证。
sudo losetup /dev/loop0 enc.disk
tpm2_startauthsession --policy-session --session=session.ctx
tpm2_policypcr -Q --session=session.ctx --pcr-list="sha256:0" --
policy=pcr0.sha256.policy

# 此时，理想情况下，您希望在内存中解开秘密，并将其直接管道到 cryptsetup，如下所
示:“tpm2_unseal——auth=session:session。——object-
context=0x81010002 | sudo cryptsetup luksOpen——key-
file=- /dev/loop0 encvolume”。
# 但是，为了在下一节演示灵活 PCR，我们将复制未密封的秘密:
tpm2_unseal --auth=session:session.ctx --object-
context=0x81010002 > disk_secret.bkup
cat disk_secret.bkup | sudo cryptsetup --key-
file=- luksOpen /dev/loop0 enc_volume
tpm2_flushcontext session.ctx
sudo mount /dev/mapper/enc_volume mountpoint/
ls mountpoint/

# 为了防止进一步开封，PCR0 将被延长。这将导致 PCR0 保持不同的值，就像在固件替换攻击期间
一样。这将导致策略检查失败，从而导致打开尝试失败。
# 延长前观察 PCR 状态，延长后再次观察:
tpm2_pcrread --sel-list=sha256:0
tpm2_pcrextend 0:sha256=0000000000000000000000000000000000000000000000000000000000000000
tpm2_pcrread --sel-list=sha256:0

# 尝试用脏 PCR 打开密封的磁盘加密密匙:
```

```
tpm2_startauthsession --policy-session --session=session.ctx
tpm2_policypcr -Q --session=session.ctx --pcr-list="sha256:0" --
policy=pcr0.sha256.policy

# 以下操作将导致策略检查失败，从而阻止开封操作：
tpm2_unseal --auth=session:session.ctx --object-context=0x81010002
tpm2_flushcontext session.ctx

# 卸载磁盘：
sudo umount mountpoint
sudo cryptsetup remove enc_volume
sudo losetup -d /dev/loop0
```

该方案检索 LUKS 加密密码口令之前通过使用 TPM PCR 验证系统状态。但是现在有一个新问题:更新。在本例中，对系统应用固件更新可能导致加密数据无法访问，因为 PCR0 表示固件的特定版本。为了成功地更新系统，有必要根据一组预测的 PCR 值重新密封秘密。但是，如果回滚更新，旧的值将不再工作。此外，在实际执行更新之前预测更新后的 PCR 值可能是不切实际的——最好的方法可能是在相同的系统上应用更新，看看会产生什么 PCR 值，并预先授权两组 PCR 值。对于 TPM，有一种方法可以做到这一点，称为“被授权的 PCR 策略”（authorized PCR policy）。

4.3.5 基于被授权的 PCR 策略保护带有密码口令的 LUKS

被授权的 PCR 策略作为 TPM 密封对象的身份验证机制。不再使用严格的 PCR 策略绑定到原始 PCR 值，我们现在密封它的 PCR 签名。PCR 集由系统设计人员签名，并由 TPM 进行验证。 这一方案可以通过以下步骤实现:

引入一个 RSA 公私钥对使用 OpenSSL 命令行工具生成 RSA 对。私钥将离线存储。公钥将与授权的 PCR 策略一起分发，并用作 LUKS 加密密码口令的授权机制。

```
1.  openssl genrsa -out signing_key_private.pem 2048
2.  openssl rsa -in signing_key_private.pem -out signing_key_public.pem -pubout
```

用私钥签署一组 PCRS 在每个已知的良好系统配置上，收集当前的 PCR 值并用 RSA 私钥对它们进行签名。这个步骤可以根据需要进行多次。

```
tpm2_startauthsession --session=session.ctx
tpm2_policypcr -Q --session=session.ctx --pcr-list="sha256:0" --
policy=set2.pcr.policy
tpm2_flushcontext session.ctx
openssl dgst -sha256 -sign signing_key_private.pem -
out set2.pcr.signature set2.pcr.policy
```

将 LUKS 加密密码口令绑定到公钥

```
# 在将 LUKS 加密密码口令密封到 TPM 之前，有必要创建一个策略对象，该对象指定可以解封密码口令的条件。该策略将指定一组特定的 pcr (PCR0)必须与使用特定密钥(signing_key_public.pem)签名的值匹配:
tpm2_loadexternal --key-algorithm=rsa --hierarchy=o --
public=signing_key_public.pem --key-context=signing_key.ctx --
name=signing_key.name
tpm2_startauthsession --session=session.ctx
tpm2_policyauthorize --session=session.ctx --policy=authorized.policy --
name=signing_key.name
tpm2_flushcontext session.ctx

# 通过使用上述策略创建一个密封对象，将密码口令密封到 TPM。请注意，由于前面的示例扩展了 PCR0 以防止密码口令的重新解密，因此使用了密码口令的备份副本:
cat disk_secret.bkup | tpm2_create --hash-algorithm=sha256 --
public=auth_pcr_seal_key.pub --private=auth_pcr_seal_key.priv --sealing-
input=- --parent-context=prim.ctx --policy=authorized.policy

# 用上面创建的对象替换旧的持久密封对象:
tpm2_evictcontrol --hierarchy=o --object-context=0x81010002
tpm2_load -Q --parent-context=prim.ctx --public=auth_pcr_seal_key.pub --
private=auth_pcr_seal_key.priv --name=auth_pcr_seal_key.name --key-
context=auth_pcr_seal_key.ctx
tpm2_evictcontrol --hierarchy=o --object-
context=auth_pcr_seal_key.ctx 0x81010002
```

解密加密密码口令

```
# 加载公钥、PCR 策略和签名，并要求 TPM 验证签名:
tpm2_loadexternal --key-algorithm=rsa --hierarchy=o --
public=signing_key_public.pem --key-context=signing_key.ctx --
name=signing_key.name
tpm2_verifysignature --key-context=signing_key.ctx --hash-algorithm=sha256 --
message=set2.pcr.policy --signature=set2.pcr.signature --
ticket=verification.tkt --format=rsassa

# 现在请 TPM 验证 PCR 值是否与当前值匹配，并为签名验证传递一个验证票据。请注意，一次只能验证一组 PCR 值:整个过程必须重复，以便尝试验证另一组签名 PCR 值:
```

```
tpm2_startauthsession --policy-session --session=session.ctx
tpm2_policypcr --pcr-list="sha256:0" --session=session.ctx --
policy=set2.pcr.policy
tpm2_policyauthorize --session=session.ctx --input=set2.pcr.policy --
name=signing_key.name --ticket=verification.tkt

# 解锁加密密码口令并解锁卷:
sudo losetup /dev/loop0 enc.disk
tpm2_unseal --auth=session:session.ctx --object-
context=0x81010002 | sudo cryptsetup --key-
file=- luksOpen /dev/loop0 enc_volume
tpm2_flushcontext session.ctx
sudo mount /dev/mapper/enc_volume mountpoint/
ls mountpoint/

# 卸载卷
sudo umount mountpoint
sudo cryptsetup remove enc_volume
sudo losetup -d /dev/loop0
```

4.4 基于国密的可信计算最佳实践

4.4.1 TPM、TCM 与国密算法

可信平台模块始于 2000 年可信计算平台联盟（Trusted Computing Platform Alliance）制定的 TPM 1.0 规范。2003 年，TCG(trusted computing group) 成立，修改完成了 TPM 1.1 规范，2004 年发布了 TPM 1.2，2014 年发布了 TPM 2.0 规范。

鉴于可信计算技术对国家信息安全体系的重要性，经国家密码管理局批准，中国于 2006 年成立了可信计算密码专项组，并于 2008 年 12 月更名为中国可信计算工作组(China TCM Union)，简称 TCMU。2007 年 12 月，国家密码管理局颁布了《可信计算密码支撑平台功能与接口规范》，将国内使用的可信基础模块定义为 TCM(trust cryptography module)。相较于 TPM，TCM 采用了我国《商用密码管理条例》中规定的 SM2、SM3 等国密算法，同时引入了对称密钥算法，简化了 TPM

中复杂的密钥管理。TCM 的证书认证机制采用签名密钥以及加密密钥的双证书机制，将对称密钥与非对称密钥结合保护系统安全，在密钥管理体系和基础密码服务体系等方面进行了改进，提升了系统的安全性。TPM 和 TCM 的构成和功能类似，提供可信计算平台的信任根（RTS, RTR），是由 CPU、存储器、I/O、密码协处理器、随机数产生器和嵌入式操作系统等部件组成的独立 SoC 芯片，具备可信度量的存储、可信度量的报告、密钥产生、加密和签名、数据安全存储等功能。

2015 年 TPM 2.0 library specification（Trusted Platform Module）正式成为国际标准 ISO/IEC 11889，吸纳了 TCM 中相关的安全改进，并首次成体系支持中国密码算法体系，包括 SM2/SM3/SM4 密码算法。这是中国密码算法技术和标准的又一次重要突破，也是中国信息安全标准在国际标准化工作中的重要进展。ISO/IEC 11889 支持中国商用密码算法体系（SM2/SM3/SM4），使得在数据安全保护上更加牢不可破。

4.4.2 TPM 2.0

TPM （Trusted Platform Module）2.0 是遵循 ISO/IEC 11889 系列标准的可信根、由国际可信计算组织 TCG（Trusted Computing Group）维护。如下图所示，密码学系统是 TPM2.0 核心、其所有安全功能均以密码学系统为基础。

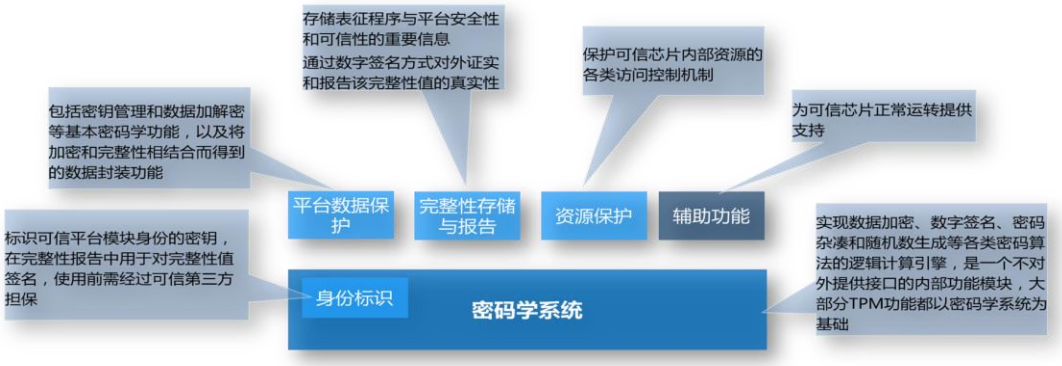


图 4-4-1 TPM 核心功能

目前 TPM2.0 支持 SM2、SM3、SM4 特性，具体包括基于 SM3 的 Hash 算法、支持 SM3 bank 的 PCR extend；支持 SM4 加解密；支持 SM2 加解密、SM2 签名验签、SM2+SM3 证书签名验签。

龙蜥 OS 为客户提供了两种使用 TPM 国密算法引擎的途径：TSS(TrustedSoftwareStack)提供的 API 访问 TPM 国密算法、TPMtools 中的密码指令使用 TPM 提供的国密算法

以下示例为使用 tpmtools 创建基于 SM2+SM3-256 的认证密钥。

```
h_ek_persistent_ecc=0x81010002

tpm2_createak -C ${h_ek_persistent_ecc} -G ecc -g sm3_256 \
    -s sm2 -c ak_ecc.ctx -u ak_ecc.pub -n ak_ecc.name -T device
```

4.4.3 UEFI 可信启动（Trusted Boot）

可信启动（TrustedBoot)是以可信根为核心、检测系统启动过程中所加载和使用的组件、确保预期的组件在预期的节点加载运行。

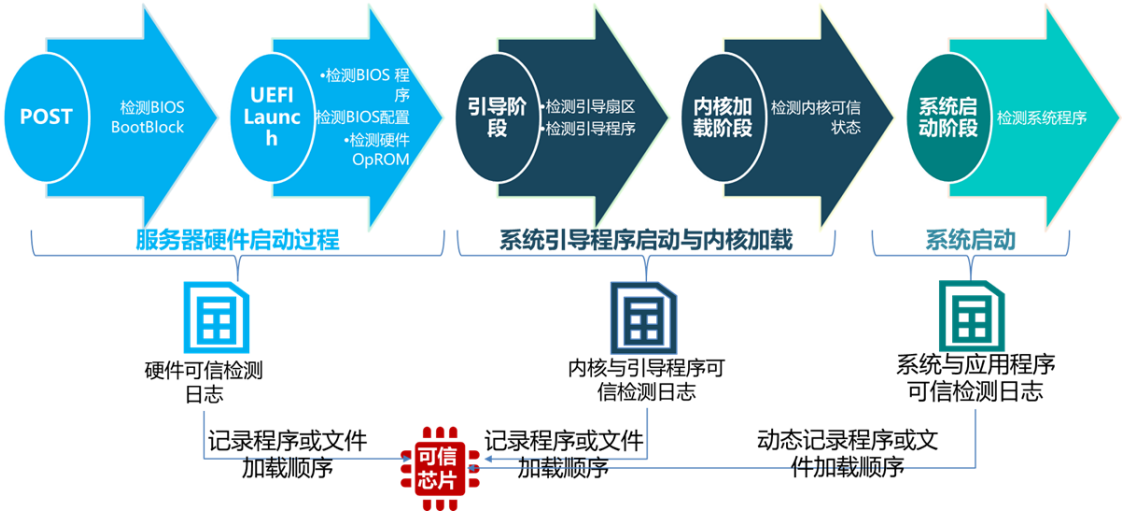


图 4-4-2 可信启动示意图

使用国密可信启动的条件如下：

- 硬件要具备的特性：
  - 服务器计算机固件（UEFI）支持基于国密的可信度量特性
  - 服务器集成了 TPM2.0 芯片（该芯片许支持 SM3、PCR SM3 Bank 等特性）
- OS 要具备的特性：
  - 部署支持基于国密可信启动的引导程序（grub）
  - 部署可信启动策略管理工具 (iTrustMidware)
  - **说明**：当以上组件可从 KeyarchOS 获取，KeyarchOS 是 基于龙蜥 OS 的商业发行版

1）BIOS 国密可信度量 Enable 方法：需要将 SM3_256 PCR Bank 改为 Enable

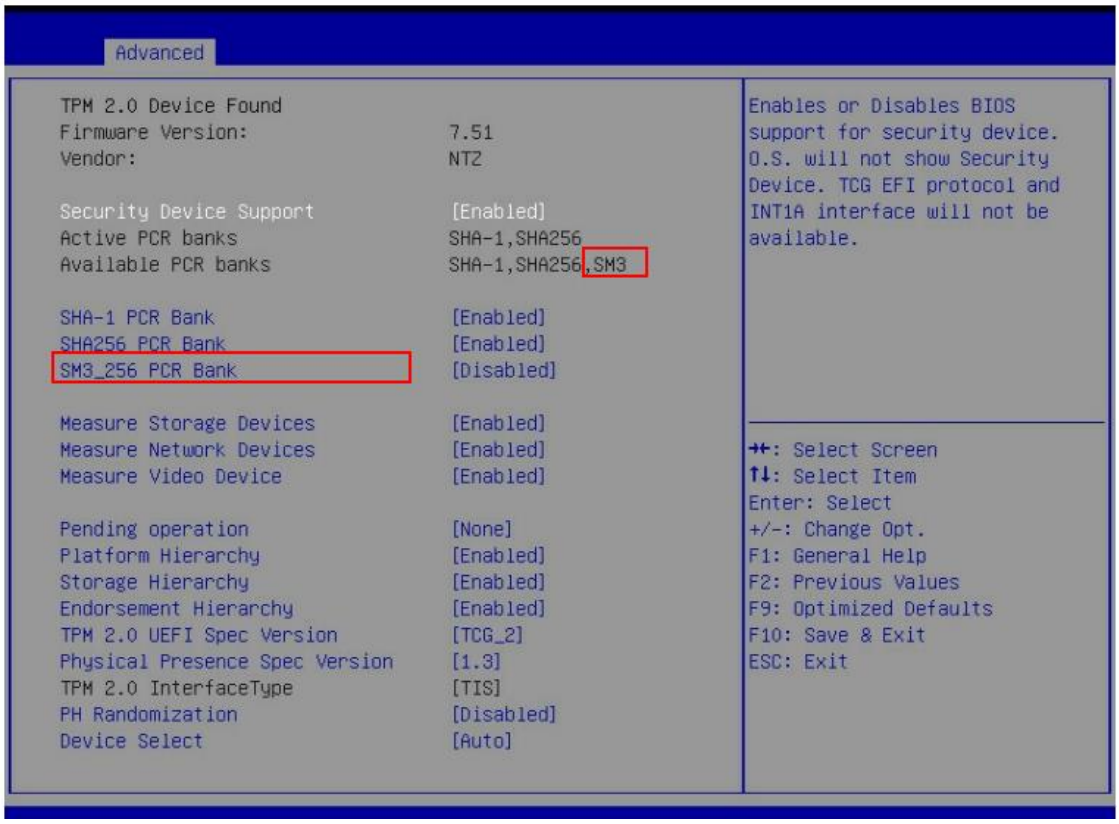


图 4-4-3 BIOS 国密可信度量算法配置

2）OS 国密可信度量组件部署方法：

```
# grub2 系列所有组件（如 grub2、grbu2-common、grub2-efi-x64 等）均需部署
rpm -ivh grub2-xxx.anolis.x86_64.rpm
rpm -ivh iTrustMidware-3.0.1-20220827200013.kos.x86_6.rpm
```

3）部署上述组件后、默认支持 kernel、initramfs、grubcmdline 的国密可信度量

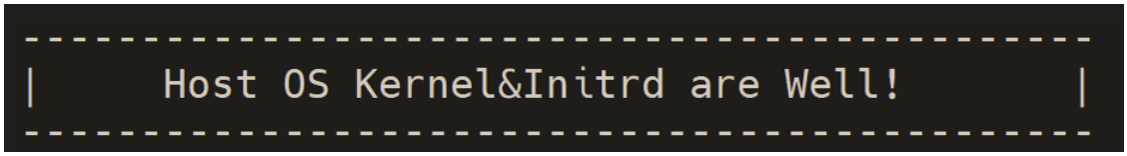


图 4-4-4 *引导阶段内核可信验证结果*

4）如需添加对特定系统程序或配置文件的度量需要进一步配置策略

```
> tlcptool
1. Check Policy State.
2. Turn on Supervisory Policy.
3. Update Supervisory Policy.
4. Turn on Interception Policy.
5. Update Interception Policy.
6. Turn off Policy.
7. Export BootLoader Passphrase.
8. Deploy Measurement File.
9. Update Measurement File.
10. Delete Measurement File.
11. Export Software Trusted Report.
O.Exit.
Please Input the Corresponding Operations.
```

5）如需添加启动控制策略（在可信度量的基础上、基于国密进一步进行可信验证、需要配置可信启动控制策略。

6）度量结果查看：直接在 /sys/kernel/security/tpm0/binary_bios_measurements 查看（以下是输出的部分片断）。

```
0 0d818d47f8b7... [S-CRTM Version]
0 b16790da86a8... [POST CODE]
7 c3e86209704b... [EV_EFI_VARIABLE_DRIVER_CONFIG] SecureBoot
... ...
8 c9eef8824efb... [IPL] grub_cmd set gfx_payload=keep
8 6a1007c86dc8... [IPL] grub_cmd insmod gzio
8 eb204c91fc3d... [IPL] grub_cmd linux (hd0,gpt2)/vmlinuz-
4.18.0 root=/dev/sda2 ro
9 3e9ff31a4687... [IPL] grub_linuxefi Kernel
8 603cfbaa8375... [IPL] grub_cmd initrd (hd0,gpt2)/initramfs-4.18.0.img
9 f0ba7132ccea... [IPL] grub_linuxefi Initrd
```

其次使用 iTrustMidware 提供的策略管理工具也可以查看到详细的度量日志。

4.4.4 TPM 国密证书部署与远程证明

远程证明是可信计算中实现节点可信认证的关键设施，是实现可信节点之间可信互联的依据。KOS（Anolis 衍生发行版）在 RSA + SHA256 + AES 和 ECDSA + SHA256 + AES 的基础上拓展支持了基于 TPM 2.0 国密的远程证明特性。

1) 在 KOS 部署可信代理端之后、配置后可在可信管理端识别到响应的节点、之后通过可信管理端可以部署相应节点的国密远程证明证书（Attestation Certification)。

2) 部署可信证书后、节点即可以用远程证明的方式向可信管理端发送报告进行可信证明。

4.5 基于机密计算的虚拟可信根解决方案

4.5.1 背景

目前，有四种类型的 TPM 广受欢迎，包括独立型（DISCRETE） TPM、集成型（INTEGRATED） TPM、固件（FIRMWARE）TPM 和软件（SOFTWARE） TPM。它们在成本、功能和安全性之间提供了不同的权衡。在虚拟化场景或云计算中，软件 TPM 更受偏爱，因为它具有更好的灵活性和易用性、成本效益以及测试和原型设计等方面的优势。然而，软件 TPM 可能更容易受到软件错误和攻击的影响，因此需要更多的组件来保持安全，包括 TEE 操作系统和运行在 TEE 中的应用程序代码。

许多云安全产品中的软件 TPM 解决方案可以为 Guest 提供可信计算的功能。这些方案基于硬件和虚拟信任根，在服务器硬件可信的基础上，将信任链从 Host 延伸到 Guest，构建了安全体系。然而，这些虚拟 TPM 方案在敏感数据安全方面无法满足租户的要求。因为 TPM 命令请求和响应的内容在链路传输中以明文形式传递，并且敏感操作结果保存在 Host 侧的软件 TPM 后端服务实例中，这导致租户保存在软件 TPM 的敏感数据容易受到攻击和泄露，无法保证数据的机密性。

Confidential vTPM 方案可以解决上述问题，它旨在利用 TEE 技术加强对软件 TPM 后端设备实例的保护能力。即使在最坏情况下，如 Host 环境遭到攻击或基础

设施 owner 有意泄露租户数据，租户的敏感数据也能得到保护，不会被泄露。Confidential vTPM 方案的目标是部分消除租户担心软件 TPM 后端设备实例泄露敏感数据并进一步导致数字资产遭泄露的安全风险。

项目地址：https://github.com/inclavare-containers/confidential-vtpm

4.5.2 可信计算与机密计算

区别与联系

可信计算和机密计算（TEE）是两种不同的安全技术，但它们之间存在一定的关系。

可信计算旨在确保计算过程的完整性和可信度。它通过硬件保护和隔离机制，如可信平台模块（TPM）、安全处理器等，保护计算过程中的敏感信息和代码免受恶意攻击和篡改。可信计算通常用于保护整个计算平台的完整性和安全性。

机密计算（TEE）旨在确保敏感数据和代码在计算过程中的保密性。TEE 通过创建一个安全的执行环境，保护敏感数据和代码免受攻击者的获取和篡改。它通过硬件隔离和加密技术，确保敏感计算在受到高级攻击者的威胁时仍然得到保护。

TEE 和 TPM 在度量和认证方面有相似之处，但它们之间最大的区别在于 TEE 提供的安全隔离性能够缓解甚至消除来自 Host 侧的攻击面，使得 TEE 内暴露的外部服务（如 App 服务）成为最显著的攻击面。相比之下，基于 TPM 的可信系统攻击面要大得多。因此，TEE 技术的价值在于它对攻击者造成的破坏性做了最大程度的预期，安全水平要远高于基于 TPM 的可信系统。

基于 TPM 的可信系统需要在软硬件上增加很多额外的防护机制以实现纵深防御，增加攻击者渗透Host的难度，同时通过在运维层面增加很多安全流程尽可能防止内

鬼。而 TEE 则是在这些防护和流程被攻破或绕过时，建立安全假设的。因此，如果用户不需要如此高的安全水平，不担心来自 Host 的安全威胁，那就不必使用机密计算 TEE 技术。

总体来说，TEE 技术提供了与基于 TPM 的可信系统相比更高级别的安全保护。虽然这种技术的实现方式不同，但其价值在于保护敏感数据和代码免受攻击者的获取和篡改。因此，在选择何种技术时，需要根据实际需求和安全风险进行综合考虑。

相互结合

机密计算（TEE）技术的出现为应对当前日益增长的安全问题提供了一种全新的解决方案，然而，它并不能完全取代 TPM 技术。这是因为 TEE 技术具有以下三个限制：

1) TEE 技术一般度量的寄存器并不多，不能提供细粒度的度量。相比之下，TPM 技术可以提供更加精细的度量记录，以便进行更为细致的验证和认证。
2) 现有的 TPM 的上层生态已经相当成熟，可以复用软件栈。与之相比，TEE 技术的生态尚不够成熟，需要更多的研究和开发工作才能达到同样的成熟度。
3) 套用 TPM 的远程证明流程可以解决异构 TEE 的机密互联的问题。TPM 技术已经具备在异构计算平台上进行远程证明的能力，并且已经有一套成熟的流程可以进行实施。这意味着，即使使用不同的 TEE 设备，也可以通过套用 TPM 的远程证明流程来解决 TEE 之间的机密互联问题。

为了能够更好地将可信计算技术与机密计算技术相结合使用，建议在 TEE 中使用可信计算技术，在可信主机执行环境中使用机密计算技术。通过 Confidential vTPM 项目，可以为租户提供安全等级更高的云主机环境。这种结合使用可信计算

技术和机密计算技术的方案，可以在保护计算过程的同时，保护租户的敏感数据和机密计算任务的安全性。

4.5.3 架构

Confidential vTPM 的解决方案由四个部分组成：前端、后端、认证服务和可信 CA。认证服务和可信 CA 是运营商或用户部署的第三方可信服务，用于提供数字身份验证和认证，确保用户身份的真实性和数据的机密性。

后端是一个 TEE 化的 swtpm/swtcm，要求进程运行在 TEE 内。是负责处理软件 TPM 计算任务的核心组件，它提供了一个安全的执行环境，可以确保敏感数据和代码在计算过程中的机密性和完整性。

前端由是负责提供虚拟化环境的组件，它可以启动机密虚拟机或普通虚拟机。虚拟机内的进程可以运行在 TEE 内，以提供更高级别的安全保护。

TPM attestation 服务和可信 CA 是可以由运营商或用户部署的第三方可信服务（若用户不相信运营商可由自己部署）。这些服务可以提供数字身份验证和认证，以确保设备或用户身份的真实性和数据的机密性，并提供可信计算的证明。

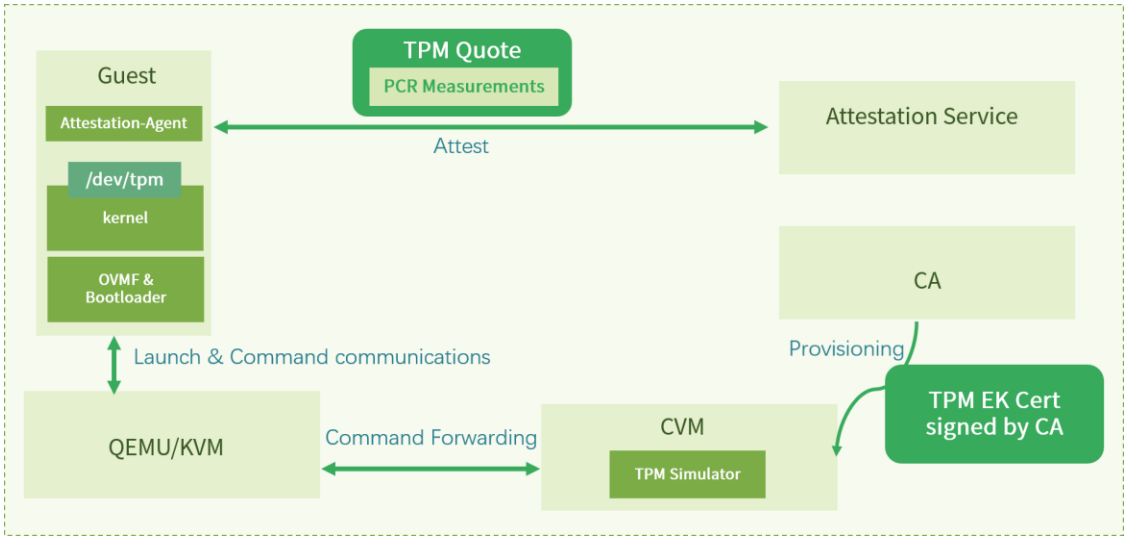


图 4-5-1 Confidential vTPM 架构图

- 在前端处理过程中，采用了虚拟 I/O (virtio) 技术，对涉及到可信平台模块 (TPM) 消息传递的 qemu/edk2/kernel 组件进行了增强。这种改进确保了来自 Guest OS 的消息在传递到 qemu 之前已经被加密，从而提升了消息的机密性。
- 在后端处理过程中，将软件 TPM (swtpm) 部署在具有安全保护的可信执行环境 (TEE) 中。这种部署方式确保了敏感信息在后端设备实例中的机密性。通过引入了第三方证书颁发机构 (CA) 来管理虚拟 TPM EK 证书，提高了证书的合法性和可信度。
- 通过重新设计一套认证注册流程，该流程不仅验证了平台是否受到 TPM 的保护，还验证了平台是否运行在 TEE 环境中。通过这个流程，可以为实现多平台 TEE 的机密互联打下了坚实的基础，提供了更高的安全性和可信度。

4.5.4 安全特性

- 使用 Enclave-CC 部署 swtpm 后端，实现高效、安全和可扩展的大规模部署。Enclave-CC 具有 K8s 无缝对接能力和硬件安全扩展功能，可轻松管理敏感工作负载，并为用户提供与普通容器相媲美的使用体验。
- 采用可信任的安全数据存储方案，将 swtpm 的非易失存储内容妥善存储在一个经过严格验证的可信任数据库中。只有在可靠的信道下，才会从数据库中读取这些宝贵数据信息，以确保数据的安全性和完整性。
- 建立机密的数据传输信道，利用机密计算远程证明技术和成熟的 TLS 安全信道传输协议相结合的 RATS-TLS 库，为用户提供更加安全可靠的跨平台机密计算服务，确保用户数据得到最高级别的安全保障。
- 在龙蜥社区开源 Confidential vTPM 方案。通过开源，将能够更快地推进项目的发展，并为用户提供更加可靠和安全的 TPM 解决方案。

4.6 可信计算 3.0 解决方案

4.6.1 可信计算 3.0 系统架构

系统架构概述

可信计算 3.0 双体系可信计算架构是整个主动免疫防御体系的基础结构保障，构建了安全隔离的计算部件与防护部件并存的双体系架构，计算部件无法访问防护部件的资源，防护部件可访问计算部件的所有资源，双方通过安全的专用通道进行交互，这 是构建整个主动免疫防护体系的基石。防护部件以可信平台控制模块 TPCM 为核心和信任源点，能够先于计算部件处理器启动，并通过直接总线共享机制访问主机所有资源，进行静态和动态可信验证，通过度量方能启动或继续执行，

否则进行报警 和控制，主动抵御入侵行为，并能实时生成主机的可信报告，上报至可信安全管理平台进行进一步关联分析。

可信计算 3.0 双体系架构的核心部件有：

- TPCM（Trusted Platform Control Module）是可信平台控制模块，是可信计算 3.0 中的一个核心组件。它不仅提供了传统的密码学功能和度量功能，还提供了主动度量和主动防御功能。TPCM 可以主动监 控和控制计算平台的运行状态，实现对计算平台的主动免疫防护。

- TSB（Trusted Software Base）是可信软件基，是可信计算平台的可信性提供支持的软件元素的集合。TSB 是可信计算 3.0 的基础，它通过自身的完整性保护、隔离保护和密钥管理等机制，保证了自身的可信性，并为 上层应用提供了可信接口。

- TMC（Trusted Management Center）是安全管理中心，它是可信计算 3.0 体系架构中的重要组成部分，负责对可信计算环境进行统一的安全策略管理和控制。它是逻辑上独立的可信子系统，通过密码学和硬件安全技 术构建完整的信任链条，通过安全管理应用，实现对网络信息系统的可管性，支持系统管理、安全管理和审计管理等功能，并可以与其他安全管理中心相连接，形成一个分布式的安全管理网络。

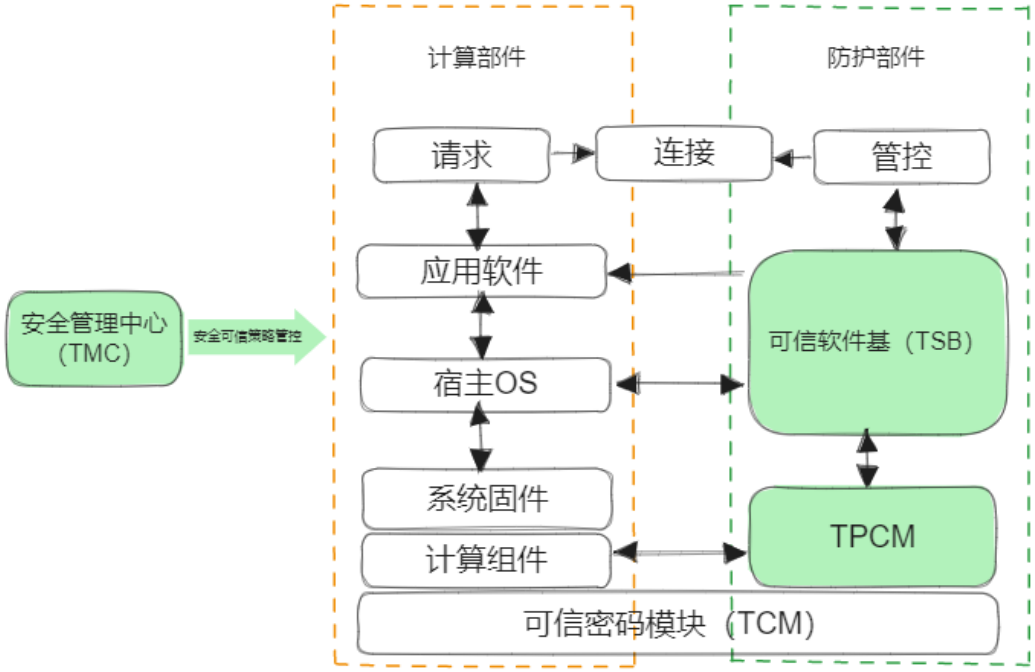


图 4-6-1 可信计算 3.0 双体系架构

可信计算节点中的 TPCM 架构[8]

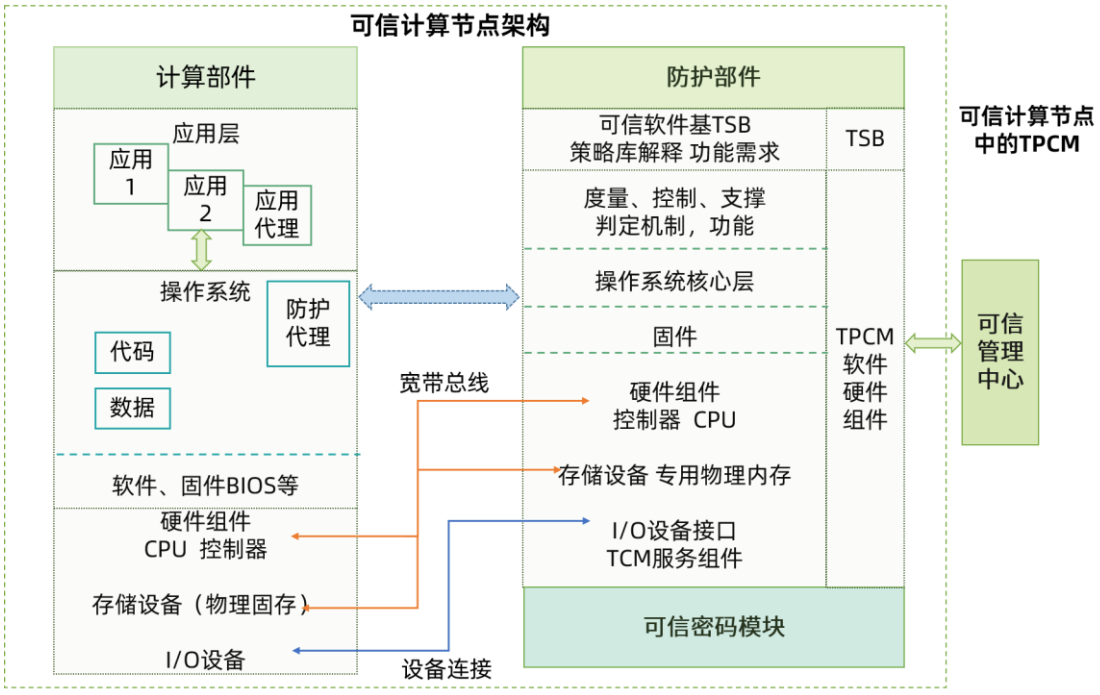


图 4-6-2 可信计算节点中的 TPCM 架构

可信计算 3.0 中的可信计算节点由计算部件和防护部件两部分组成。 计算部件是指执行计算功能的硬件和软件，包括 CPU、内存、存储设备、操作系统、应用程序等。防护部件是指负责提供可信计算功能的硬件和软件，包括可信密码模块（TCM）、可信平台控制模块（TPCM）、可信驱动模块（TDD）、可信软 件基（TSB）和可信网络连接（TNC）。

计算部件和防护部件之间通过宽带总线连接。TCM 负责生成和存储可信密钥，并对计算部件进行度量。TPCM 负责对计算部件进行控制，并对可信密钥进行保护。TDD 负责为 TPCM 提供与计算部件的接口。TSB 负责提供可信计算的软件基础， 包括可信应用程序接口（API）、可信度量库等。TNC负责提供可信计算的网络安全功能。

可信计算节点的架构主要包括以下特点：

- 双体系结构：计算部件和防护部件逻辑相互独立，形成具备计算功能和防护功能并存的双体系结构。
- 可信根：TCM 和 TPCM 构成了可信根，可信根是可信计算的核心，负责提供可信计算的功能和服务。
- 主动防御：可信计算采用主动防御的方式，通过对计算部件进行度量和控制，来识别和阻止恶意行为。

可信计算节点主要包括以下功能：

- 可信启动：可信计算节点在启动时，TPCM 会先于计算部件启动，并对计算部件进行度量和验证，确保计算部件处于可信状态。

- 可信执行：可信计算节点在运行时，TPCM 会对计算部件进行动态监控，及时发现和阻止恶意行为。
- 可信存储：可信计算节点会对敏感数据进行加密存储，防止被恶意软件窃取。
- 可信网络：可信计算节点会对网络通信进行加密，防止网络攻击。

TPCM 功能及接口框架[8]

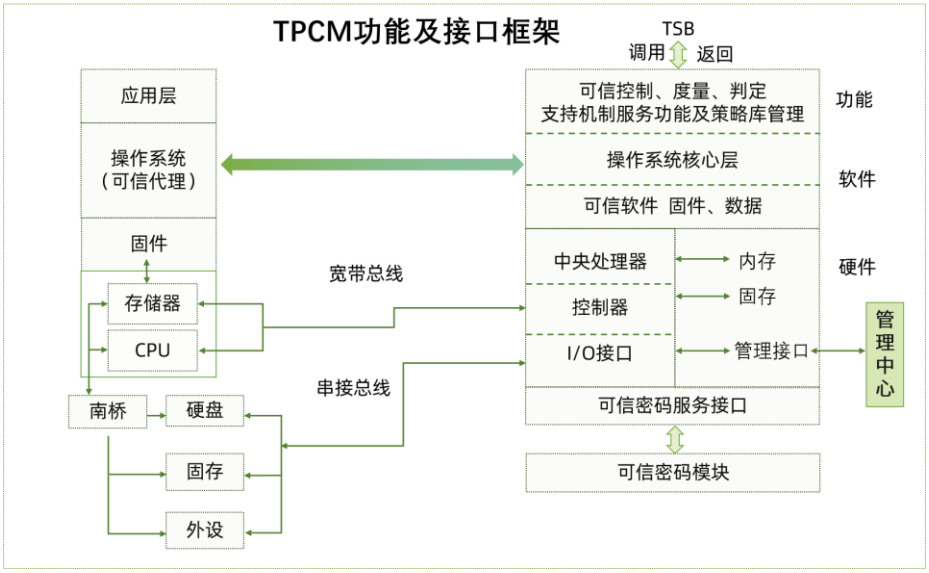


图 4-6-3 TPCM 功能及接口框架

图 4-6-3 中的左侧是 TPCM 的功能层次结构。应用层提供可信控制、度量和判定服务，包括可信身份验证、访问控制、数据完整性保护和软件完整性保护。操作系统核心层提供操作系统可信代理（OSA），OSA 是 TPCM 的接口点。

图 3 中的右侧是 TPCM 的接口框架。TSB 是 TPCM 的调用接口，用于将应用程序与 TPCM 功能连接起来。管理接口用于管理 TPCM 组件。

4.6.2 可信计算 3.0 与计算基础设施生态的融合

为了让终端用户更好的触达和使用到可信计算 3.0 的技术，可信华泰在与计算技术设施生态的融合上做了若干实践。 可信华泰与基础设施生态的厂商，包括：CPU 平台厂商、BIOS 厂商、整机厂商、OS 厂商进行了广泛合作，做了以下的实践：

1) 在海光处理器、飞腾处理器的独立安全核心内引入 TPCM 固件，让 CPU 具备 TPCM 的能力。

2) 与 BIOS 厂商合作对 BIOS 进行可信 3.0 的改造，让 BIOS 配合 TPCM，使 TPCM 具备主动度量 BIOS、GRUB、OS Kernel 的能力。

3) 整机厂商在选择确定整机产线 CPU 平台的基础上，引入 TPCM 导入的流程，让 TPCM 作为整机的 Keypart 进行采购。

4) 与整机 OS 厂商进行合作，让 TSB（包括 TSS、TPCM 驱动）作为 OS 的可信 3.0 关键软件能够部署。

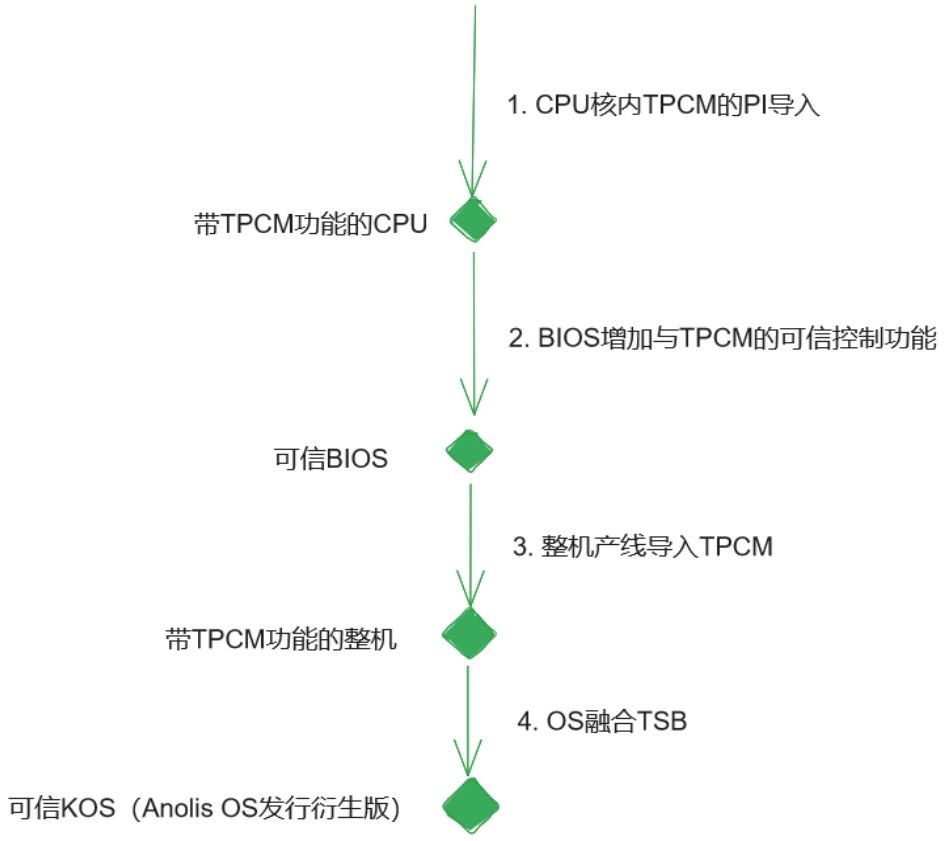


图 4-6-4 可信计算 3.0 与计算基础设施生态融合路线

与平台的融合

可信计算 3.0 在计算平台层面上，对 CPU 和 BIOS 引入了可信计算 3.0 的核心组件 TPCM、TCM，让计算平台具备可信计算 3.0 的“可信根”。

海光 CPU 核内 TPCM

注：本文以海光 CPU 作为最佳实践的举例，飞腾 CPU 目前也支持核内 TPCM，并在在架构原理上与海光 CPU 相似，仅在 TPCM 与外部通信接口不同。

在海光 2-3 号（3/5/7 全系）中的安全处理器（PSP）中植入了 TPCM 可信根固件，TPCM 固件使用密码协处理器（CCP）的 SM2/3/4 以及 TRNG 密码功能，CCP 在此处承担 TCM 的职责。

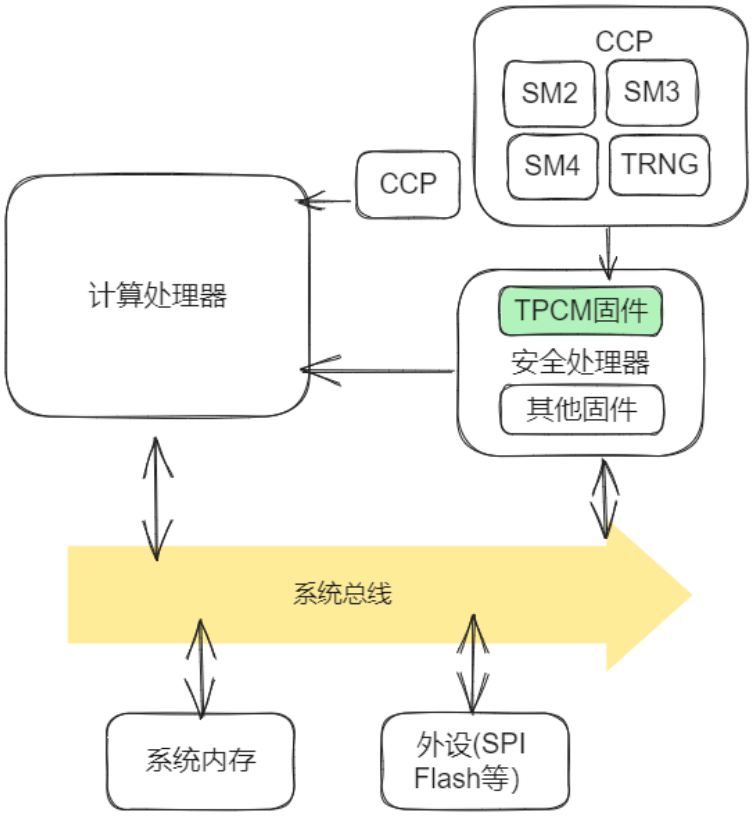


图 4-6-5 海光 CPU 核内 TPCM 架构

CPU 中嵌入 TPCM，可以让整机厂商能够随 CPU 平台选择使用可信 3.0 TPCM 作为整机 KeyPart 进行采购和产线部署。

可信 BIOS 的改造

一般 BIOS 经过可信改造，运行以下可信启动流程。

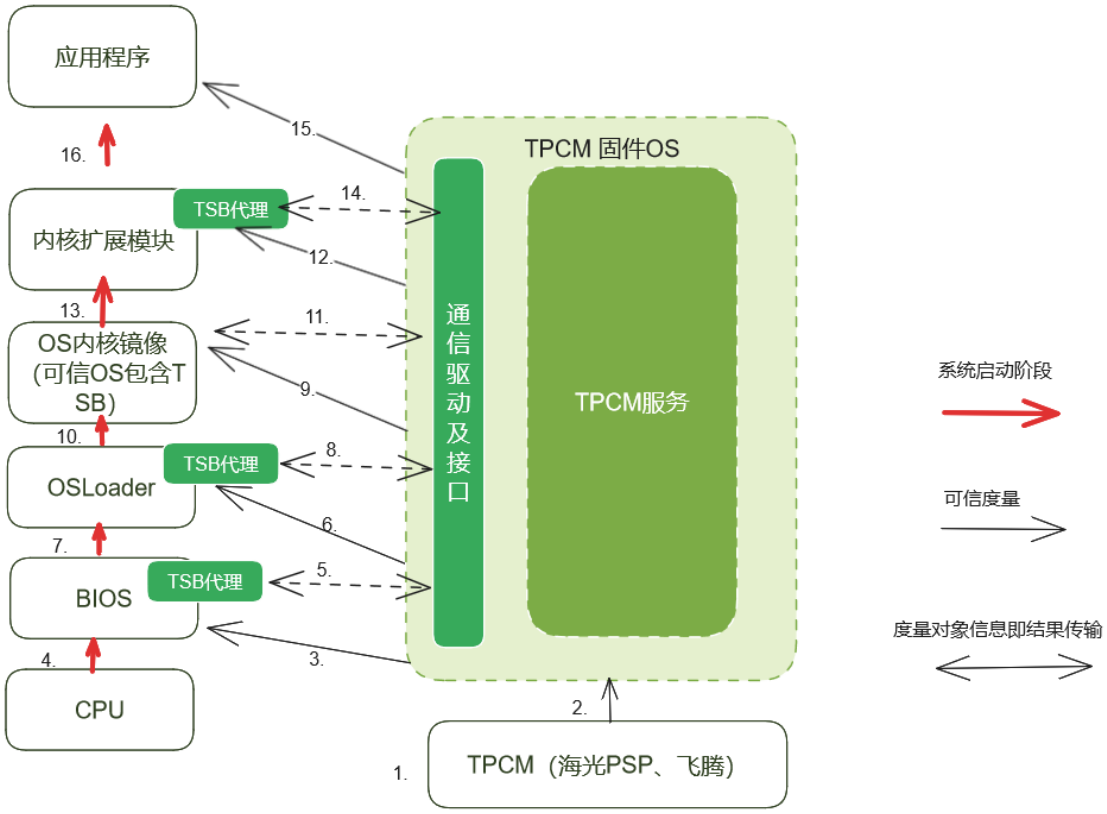


图 4-6-6 可信 BIOS

1) 设备加电，TPCM 可信根先于 CPU 启动；

2) TPCM 可信根优先启动加载可信固件，进入可信执行环境；

3) 在 TPCM 可信根的支撑下，由 BIOS 内部的 TSB 代理获取 BIOS 数据，并通过通信驱动及接口将数据传至可信根，TPCM 可信根依据可信固件的逻辑和策略对 BIOS 进行可信度量验证；

4) 上一步可信验证通过后，由 CPU 计算部件加载启动 BIOS 程序，否则，则禁止加载；

5) TSB 代理将 BIOS 的度量信息传至 TPCM 固件 OS，并将可信根对 BIOS 的度量结果在 BIOS 进行展示；

6) 在 TPCM 可信根的支撑下，由 OSLoader 内部的 TSB 代理获取 OSLoader 文件 数据，并通过通信驱动及接口将数据传至可信根，TPCM 可信根依据可信固件的逻辑和策略对 OSLoader 进行可信度量验证；

7) 上一步可信验证通过后，由 CPU 计算部件加载启动 OSLoader，否则，则禁止加载；

8) TSB 代理将 OSLoader 的度量信息传至 TPCM 固件 OS，并将可信根对 OSLoader 的度量结果 OSLoader 进行展示；

9) 通过 TPCM 可信根获取操作系统内核镜像文件数据，依据可信固件的逻辑和策略对操作系统内核进行可信度量验证；

10) 上一步可信验证通过后，由 CPU 计算部件加载启动操作系统内核， 可信软件基 TSB 随之启动，否则，则禁止加载；

11) TSB 代理将操作系统内核镜像文件数据的度量信息传至 TPCM 固件 OS，并将 TPCM 可信根对操作系统内核镜像文件数据的度量结果在 OSKernel 进行展示；

12) 通过 TPCM 可信根获取内核扩展模块文件数据，依据可信固件的逻辑和策略对内核扩展模块文件进行可信度量验证；

13) 上一步可信验证通过后，由 CPU 计算部件加载启动操作系统内核扩展模块文件数据，TSB 代理也随之同时启动，否则，则禁止加载；

14) TSB 代理将操作系统内核可扩展模块文件数据的度量信息传至 TPCM 固件 OS，并将可信根对操作系统内核可扩展模块的度量结果在内核扩展模块进行展示；

15) 通过 TPCM 可信根获取应用程序文件数据，依据可信固件的逻辑和策略对应用程序进行可信度量验证；

16) 上一步可信验证通过后，由计算部件加载执行应用程序，至此系统进入一个可信的启动环境。

与整机的融合[8]

注：可信整机产线流程目前已经在浪潮服务器、新华三服务器产线上进行了实践。

可信改造后的 BIOS 同样作为整机厂商随 CPU 平台采购的 keypart 在整机产线中进行导入。整机厂商在整机产线上导入带有 TPCM 可信根功能的 CPU 和 BIOS。

与 Anolis OS 的衍生发行版之一 KeyarchOS 的融合

KeyarchOS 是浪潮信息基于 Anolis OS 发布的服务器操作系统，**一方面，践行以系统设计为中心的技术路线，持续推动软硬协同创新；另一方面，通过 KeyarchOS 充分发挥芯片、板卡和服务器的创新成果，为用户提供卓越的整机系统体验[9]** 。可信计算 3.0 作为一种包含芯片、整机、操作系统的安全防护体系，已经 KeyarchOS 上进行了兼容性适配认证。

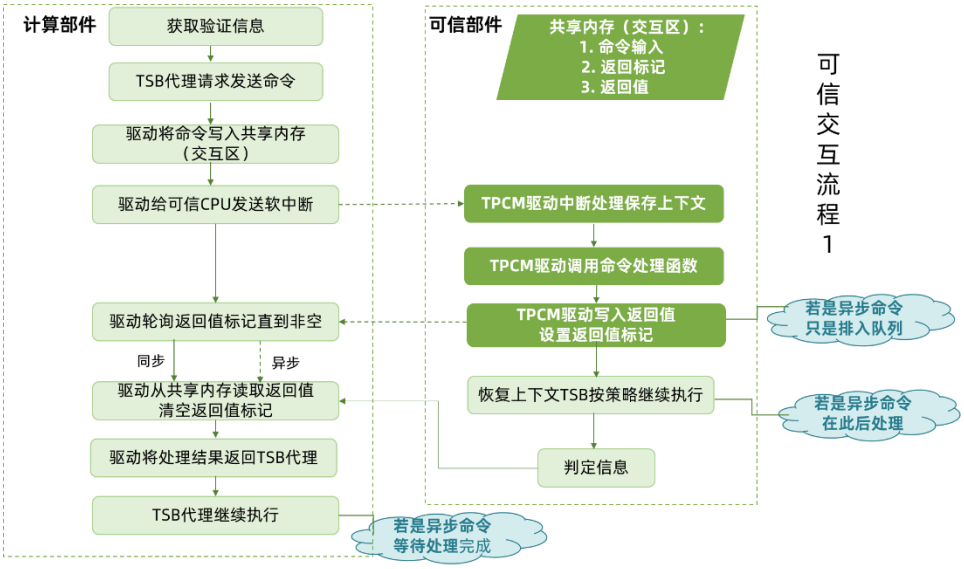


图 4-6-7 可信交互流程-*TPCM 与操作系统*

操作系统通过如图 4-6-7 所示方式与 TPCM 进行交互，计算部件和可信部件通过共享内存进行交互。可信交互流程可以是同步的，也可以是异步的。在异步交互中，可信部件会将命令排入队列，并在处理完成后返回结果。

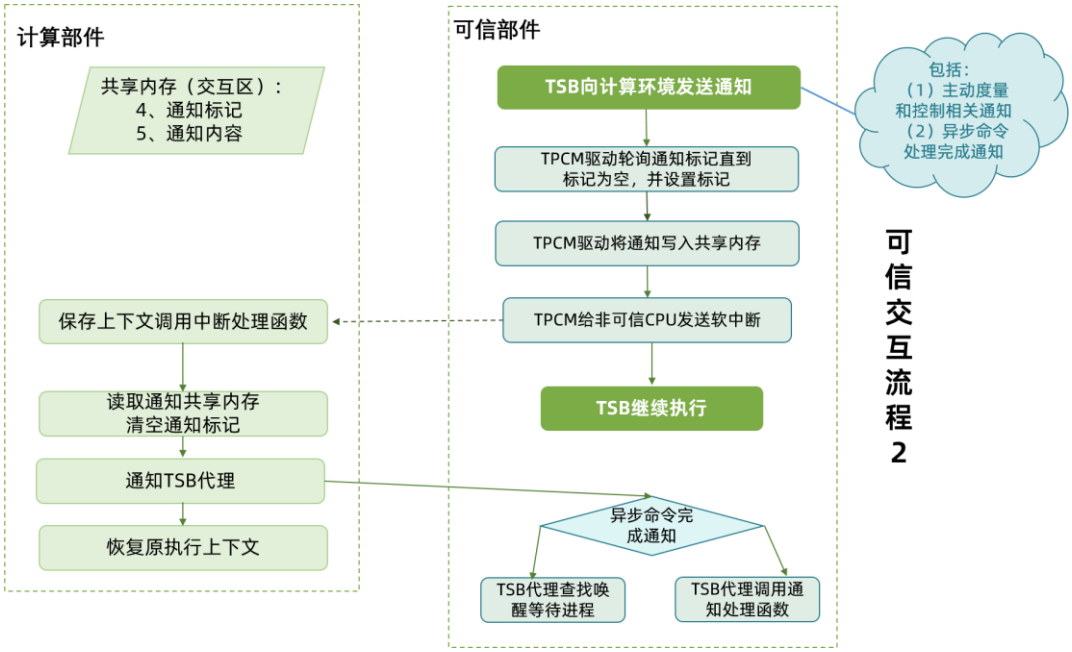


图 4-6-8 *可信交互流程-TSB 与操作系统*

图 4-6-8 是 TSB 与操作系统之间的交互流程，TSB 向计算环境发送通知。通知可以是主动度量和控制相关的，也可以是异步命令处理完成的通知。通知是通过共享内存传输的。计算环境读取通知并清空通知标记。然后，计算环境将通知传递给 TSB 代理。TSB 代理查找唤醒等待进程，并调用通知处理函数。

参考资料

[1] 冯登国,刘敬彬,秦宇等.创新发展中的可信计算理论与技术[J].中国科学:信息科学,2020,50(08):1127-1147.

[2] 沈昌祥,公备.基于国产密码体系的可信计算体系框架[J].密码学报, 2015,2(05):381-389.DOI:10.13868/j.

[3] 石文昌编著. 信息系统安全概论. 电子工业出版社，2014.68-71.

[4] 张焕国、赵波等著. 可信计算. 武汉大学出版社.

[5] 威尔·亚瑟（Will Arthur） / 大卫·查林纳（David Challener）等《TPM 2.0 原理及应用指南》. 机械工业出版社. 2017-10-1.

[6] 王勇,张雨菡,洪智等.基于 TPM 2.0 的内核完整性度量框架[J].计算机工程,2018,44(03):166-170+177.

[7] https://www.intel.com/content/www/us/en/developer/articles/code-sample/protecting-secret-data-and-keys-using-intel-platform-trust-technology.html.

[8] 沈昌祥院士《可信计算筑牢网络强国底座》演讲稿.

[9] 浪潮信息张东于 2022 年 KeyarchOS 发布会.

OpenAnolis

龙　蜥　社　区