华中科技大学

硕士学位论文

基于UEFI的信任链设计及TPM驱动程序实现

姓名：刘东丽

申请学位级别：硕士

专业：软件工程

指导教师：苏曙光

2011-01

摘 要

可信计算是计算机诸多领域里一个极为经典的课题之一，在保护计算机安全中起着至关重要的作用。基于 UEFI(United Extensible Firmware Interface, 统一可扩展固件接口)的模块化设计和对可信计算规范的支持，根据可信计算机系统的思想，将计算机启动初期最先执行的软件 BIOS 协同 TPM(Trusted Platform Module, 可信平台模块)芯片作为可信链的源头建立系统启动过程的可信链。由于目前 UEFI 的规范日趋成熟以及基于可信计算的 UEFI 规范的制定，而运行在计算机上的程序越来越大、越来越复杂，因此，研究如何从固件层进行安全防护很有意义。

将可信计算的思想引入到系统上电后的启动过程中，并对其建立可信链。在对系统的启动过程建立可信链时，先对可信任链建立核心度量根并将其作为信任链的源头，以此度量下一个执行过程的可信度，这种度量关系可以通过可信链的传递而一级一级保持下去，最终就可以保证整个系统在计算环境中的可信状态。结合 TPM 设备编程接口，实现了 TPM 设备驱动程序的在 I/O 端口通信和内存访问方式通信。TPM 设备驱动程序实现了在两种通信方式下设备初始化、发送 TPM 命令和 TPM 接收数据的功能。最后，验证了 TPM 设备驱动程序的执行结果和可信链的思想在 Windows 下的应用，所有的功能都正常运行。

可信链的建立实现了 UEFI BIOS 和可信计算平台思想的有机结合，TPM 设备驱动程序实现了可信链的底层之间的交互，为后续的理论研究和功能实现打下了良好的基础。

关键词：统一可扩展固件接口　基本输入输出系统　可信计算　可信平台模块

Abstract

Trusted Computing is one typical issue of computer research areas, which has been playing an important role in protection of computer safety. BIOS with the cooperation of TPM device, which is the first process after the system is power on, is regarded as the source of the trusted chain for the execution flow of system on the basis of the modular design of Extensible Firmware Interface and the support of Trusted Computing Specification, according the core theory of Trusted Computing System. Due to the development of EFI and the support of TCG specification, applications running on the computer become more and more complicated, it is significant to research on the computer protection beginning with firmware level.

With the start process of system and the theory of Trusted Computing brought in the process of system execution flow after system is power on, trusted chain for system execution flow is built. In the process of building the trusted chain for the start process of system, firstly core root of trust for measurement is built for the source of trusted chain, and then CRTM is regarded as the measurement of next execution, on the basis of above, the trusted relationship is passed on via the trusted chain. Finally, trusted status of platform is keep. Combining with TCG interface, TPM device driver program through the ways of I/O port and memory access communication is realized. TPM device driver program realizes the function of initialization of TPM device, sending command to TPM device and receiving data. At last, verification test about TPM device driver program and BitLocker are processed and the results show that the function of this system is normal.

The building of trusted chain realizes the theory of Trusted Computing combining with UEFI, TPM device driver program realizes the communication of base layer of trusted chain, which supplies good foundation for the following research and realization.

Key words：UEFI BIOS Trusted Computing Trusted Platform Module

独创性声明

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

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

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

本论文属于　保密□，　　在_____年解密后适用本授权书。
　　不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

1 绪论

1.1 研究的背景和意义

从 1981 年第一台个人电脑诞生开始，计算机的安全问题就一直是从业人员希望解决的问题。威胁着计算机安全的因素有很多，包括人为因素、网络因素、系统漏洞等。伴随着计算机上的软件越来越复杂，运行在计算机中的程序的越来越大，对安全问题的解决显得尤其重要。如何保障计算机中庞大的数据不丢失或不被窃取，是任何计算机应用部门首先要解决的问题。传统的利用杀毒软件、防火墙之类的保护不能从根本上保障终端平台的数据安全，究其原因都是由于平台的硬件结构比较简单，对授权用户的访问没有有效的访问验证，对执行代码不检查一致性所造成的。因此，计算机的安全需要延伸至固件甚至硬件层水平，从源头来防范不安的因素的入侵。可信计算的思想便由此产生。

可信计算（Trusted Computing, TC）是由可信计算机组（Trusted Computing Group, TCG）推出的一项新技术。它的核心思想就是在可信计算机系统中引入一个可信的第三方 TPM(Trusted Platform Module, 可信平台模块)芯片[1]，通过对第三方的度量建立一个一级一级认证的信任链，最终将这种信任扩大到整个计算机系统。

BIOS(Basic Input Output System, 基本输入输出系统)作为计算机启动初期最先执行的一个软件，负责计算机底层硬件和上层操作系统之间的联系[2]，它对整个计算机系统的安全起着至关重要的作用，是 TCP(Trusted Computing Platform, 受信任计算平台)架构可信的基础。而传统的 Legacy BIOS 从诞生之日起，二十多年来一直沿用同一个模式。在其一成不变的同时，计算机的软硬件都在进行着巨大的革新，传统的 Legacy BIOS 在可维护性、可扩展性上都面临着巨大的挑战，并成为制约计算机发展的重要因素。一种替代 Legacy BIOS 的升级方案的诞生显得尤为重要。

UEFI(United Extensible Firmware Interface，通用可扩展固件接口) BIOS 作为传统 Legacy BIOS 的一种替代方案，它定义了操作系统和平台固件之间的接口标准，是运行在操作系统和硬件之间的一个新的模型[3]。它摈弃了传统的 BIOS 的汇编语言实现，

采用 C 语言实现，使之实现了许多丰富的功能。UEFI 采用模块化的设计，在可扩展性上有了很大的改善，它提供了可编程的开放的接口[4]，主板厂商可以根据自己的实际需求，将需要的功能模块挂在 UEFI BIOS 之上。而正是由于这种开放接口，使得可信计算技术在 UEFI BIOS 框架之下得以实现。

1.2 国内外发展现状

随着计算机和网络技术的迅速发展和广泛应用，BIOS 和可信计算在这种大趋势下也得到了良好的发展。

1.2.1 可信计算的发展现状

1983 年，全球第一个《可信计算机系统评价标准》(Trusted Computer System Evaluation Criteria, TCSEC)由美国国防部颁布出台。该标准成为了首次推出可信计算机(Trusted Computer, TC)和可信计算基(Trusted Computing Base, TCB)概念的规范。1984 年，美国国防部又相继发布了《可信数据库解释》(Trusted Database Interpretation, TDI)和《可信网络解释》(Trusted Network Interpretation, TNI)。至此，标志着可信计算的初现。

1999 年，由一些著名的 IT 公司(包括微软、英特尔、惠普等)发起成立了可信计算平台联盟(Trusted Computing Platform Alliance, TCPA)。可信计算联盟的成立，标志着可信计算进入了一个快速发展的阶段。2003 年，又有几家公司(包括 Nokia, Sony 等)加入进该联盟中，改组为可信计算机组(Trusted Computing Group, TCG)。至此，可信计算技术被有效引入到应用之中并得到快速的发展。可信计算机组已经制定了相关的一些技术规范(包括可信网络、TCP 和可信存储等)。2006 年，欧洲启动了一个叫做“开放式可信计算”(Open Trusted Computing, OTC)的可信计算研究计划。

可信任计算机组织制定了可信任平台模块(Trusted Platform Module，TPM)规范，符合这种规范的芯片就是 TPM 芯片。TPM 芯片是可信计算体系中的核心，它主要完成以下功能：BIOS 开机密码加解密，硬盘加解密，完整性度量和身份验证[5]。

我国在可信计算技术领域的起步相对较晚，但是国家和各地方政府一直投入资金鼓励发展可信计算技术，带头制定了一系列的可信计算技术规范，高等院校和科研院所也在可信计算技术方面取得了长足的进步。2007 年，我国的“中国可信计算联盟”成立了。这些都标志着我国在可信计算的领域已经具备世界一流水平。

1.2.2 BIOS 及 UEFI 的发展现状

从 1981 年第一台 PC 机的诞生开始，二十多年来传统 BIOS(Legacy BIOS)一直沿用着同一个模式，随着计算机软件硬件的飞速的发展，Legacy BIOS 已经渐渐成为制约计算机向前发展的瓶颈。

2001 年，全球最大的半导体芯片制造商英特尔率先发布了替代 Legacy BIOS 的新标准——可扩展固件接口(Extensible Firmware Interface，EFI)，用以使 BIOS 的开发规范化。2005 年，由 Intel，Microsoft，AMI 等几个大厂联合组成的统一可扩展固件论坛(United Extensible Firmware Interface Form，UEFI Form)，提出通用可扩展固件接口标准[6]。

UEFI BIOS 在基本功能上与传统的 Legacy BIOS 基本一样的，都是作为底层硬件和操作系统之间的衔接桥梁。只是 UEFI BIOS 在完成平台初始化功能上更加便利，功能更加强大。UEFI BIOS 内置图形驱动，使得它可以支持鼠标图像图形界面的操作[7]。用户可以通过修改或安装新的驱动，来满足需求。可扩展的功能是 UEFI BIOS 的一个重要的功能。由于它采用模块化的设计，用户可以根据自己的实际需求将所需要的模块挂到 UEFI 上。

UEFI BIOS 的功能增强的同时也引入了不安的因素，可信任计算机组织在 2006 年提出了基于 UEFI 的受信任计算接口标准[8]。这使可信计算在 UEFI BIOS 下的实现有据可循。

我国在 BIOS 的发展上起步较晚。2005 年，信息产业部与英特尔就联合推动新一代国产 BIOS 产业项目达成意向并签署合作协议。英特尔首次授权国内厂商研发新一代国产 BIOS 产品。2006 年南京百敖软件有限公司应运而生。在国家有关部门和地方政府的支持和关怀下，历经两年多的不懈努力和实践，百敖软件解决基于 EFI 架构国产 BIOS 的技术障碍，获得了七项具有自主知识产权的国产 BIOS 相关的软件著作权，从而实现了国内 BIOS 系统从无到有的突破。2007 年 8 月，百敖软件独立研发的国内第一款安全 BIOS 产品成功地在长城计算机公司推出的“第二代安全电脑”上得到了充分的检验。国内首个安全 BIOS 产品的成功验证，使我国在信息安全的防护扩展到 BIOS 水平，具有重要的意义。

1.3 本论文主要工作及章节安排

面对目前计算机上普遍存在的安全问题，本文介绍了在基于 UEFI BIOS 的平台上引入可信计算的思想，建立了可信 UEFI BIOS 下的信任链。在详细介绍了可信任计算模型所依赖的相关技术的基础上，实现了信任链核心部件可信平台模块 TPM 的底层驱动程序，并对实现做出了验证，在 Windows Server 2008 中通过其提供的数据保护功能验证了整个可信链。

本文分为以下五个部分，具体结构如下：

第一章主要介绍了本课题的研究背景，分析了国内外在可信计算领域和 BIOS 领域的发展现状。

第二章对文章的研究对象依赖的相关技术进行了分析，包括 UEFI BIOS 的框架结构，运行阶段分析；可信计算平台组成及其核心部件，平台可信的度量机制；BitLocker 驱动加密器的体系结构。

第三章首先介绍了可信链和平台可信度量机制的概念，在这个基础上提出了 UEFI 的可信链的设计，设计了整个平台在启动过程中完整性度量如何完成。

第四章在了解了相关知识的基础之上介绍了基于 UEFI2.0 规范的 TPM 设备驱动程序的设计与实现。具体介绍 TPM 设备编程接口以及 TPM 和 BIOS 之间如何进行通信。

第五章从测试的目的、环境、步骤等方面对 TPM 设备驱动程序进行了验证。在基于 UEFI 的平台上引入可信平台模块 TPM 和可信操作系统，验证了 BitLocker 驱动加密器的功能。

第六章对本文所研究的工作进行总结和展望，并提出系统需要进一步改进的地方和未来的研究方向提出了自己的观点。

2 相关技术介绍

在将可信计算体系的思想引入到系统的启动过程中之前，首先需要了解 UEFI BIOS 的相关知识和可信计算平台的结构组成。

2.1 UEFI BIOS 技术

BIOS 是计算机上电之后执行的第一个软件，它在计算机系统中起到初始化 CPU、芯片组、内存、外围设备等部件的作用，使得它们处于一个已知的、可操作的状态，随后将控制权移交给操作系统[9]。

在计算机硬件快速发展的过程中，BIOS 维持原有的模型架构长达二十多年，传统的 BIOS 在很多方面已经成为制约计算机发展的重要因素。为了解决传统 BIOS 面临的一系列问题，人们开始提出新的 BIOS 标准或架构来解决这些问题。可扩展固件接口(Extensible Firmware Interface，EFI)技术就是在这样的背景之下产生和发展的。EFI 是一个开放的接口定义，本质上来说它是定义了平台固件和操作系统之间的接口规范[10]，不依赖于特定的 BIOS 实现。尽管 EFI 是英特尔为自己的架构而设计的规范，但是由于它的开放性和规范性，EFI 在业界得到了广泛的认可，被普遍认为是替代传统 BIOS 的下一代 BIOS 标准。同时，业界各大芯片、设备、操作系统厂商在英特尔 EFI 的规范和成果的基础上组建了 UEFI 联盟，成为了推动 EFI 发展的国际组织。现在 UEFI 的最新版本已经到 UEFI2.3 版本了。本文所说的 UEFI 除了特别指明之外都是代表 UEFI 规范而非国际组织。

2.1.1 UEFI BIOS 的框架结构

UEFI BIOS 是系统启动过程中可以使用的接口和规范。下图 2.1 反应了兼容 UEFI 标准的系统的不同组件之间的交互。UEFI BIOS 的设计主要基于以下四个要素：现存架构标准的复用(Reuse of existing table-based interfaces)、系统分区(System partition)、启动时服务(Boot services)、运行时服务(Runtime services)[11]。UEFI 启动时服务所调用的函数的接口都是 64 位的，这提高了可扩展性。在预启动环境中，UEFI

映像调用启动时服务接口函数。UEFI 运行时服务的接口函数可以在预启动的过程中被调用，也可以在操作系统运行的过程中被调用[12]。它主要提供操作系统平台硬件资源统一的接口。

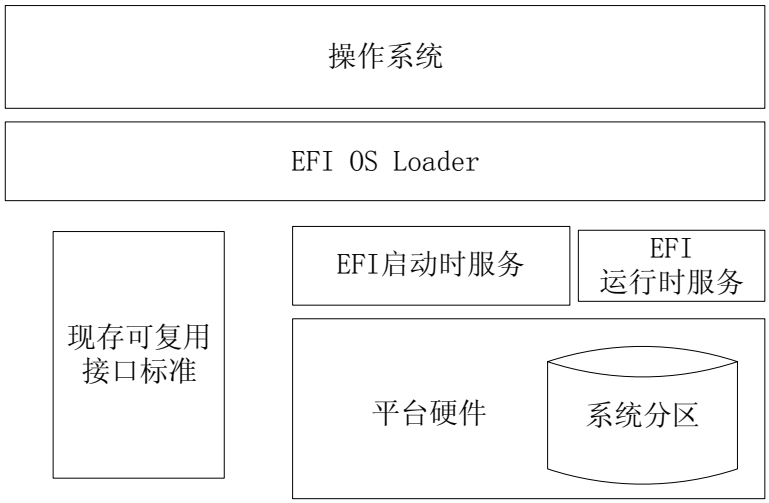

图 2.1　UEFI BIOS 的基本结构图

（1）现存可复用的接口标准

为了维持现存的在 OS 和固件之间的公共基础标准，通用可扩展固件接口标准也需要兼容一些现存的应用在平台上的接口规范。例如：高级配置与电源管理接口标准(Advanced Configuration and Power Management Interface, ACPI)和 SMBIOS(System Management BIOS)标准。

（2）系统分区

系统分区定义了允许多个供应商为了安全共享的分区和文件系统。这种可以包含一个分离的、共享的系统分区的能力，表明了在不需要过多的增加平台非易失性存储上的需求就可以提高平台的附加值。

（3）启动时服务

在启动的过程中，设备和系统可以调用的函数由启动时服务提供。设备的使用是通过对句柄和协议的抽象而实现的。这样的方法使得 UEFI BIOS 代码的复用性得到了很大的提高。

在启动过程中，通过对 UEFI 启动时服务的接口函数的调用完成对平台固件资源的统一管理。可供调用的 UEFI 启动时服务接口函数分为两种：基于句柄的和全局的。

所有的系统服务都可以被全局函数存取，并能够适用于所有的平台，通用性良好。这是因为每一个平台都支持系统服务。而对基于句柄的函数的访问时针对某些特定的设备或者是设备特定的功能，因此它只是在某些平台上可以使用，而在没有特定设备的平台上不能使用。

操作系统载入程序是 UEFI 的应用程序，它的执行必须依靠 UEFI 启动时服务来完成设备存取和内存的分配。在起始阶段，通过 UEFI 映像获取 UEFI 系统表的指针。UEFI 系统表中包含有 UEFI 启动时服务分配表和对控制台完成存取功能的默认句柄。而所有的 UEFI 启动时服务函数，都只能在操作系统载入程序之后操作系统获得控制权之前被使用[13]。一旦 UEFI 启动时服务 ExitBootServices 后，就终止了 UEFI 启动时服务。这时，只能调用 UEFI 运行时服务。

（4）运行时服务

运行时服务是为了保证系统启动到 OS 所需的基本硬件资源的抽象。和 UEFI 启动时服务所不同的是，UEFI 运行时服务的接口函数既可以在系统预启动的过程中被调用，也可以在操作系统运行的过程中被调用[14]。

无论在什么情况下，UEFI 运行时服务所占用的系统内存都将被保留下来，操作系统不能使用这部分内存。而这些内存对 UEFI 的函数来说一直可用的。UEFI 规定了 UEFI 运行时服务可以调用的系统硬件资源。所以当这些系统硬件资源被分配好时，操作系统同时得知了这一消息，就能够保证这些系统硬件资源不被操作系统过占用。

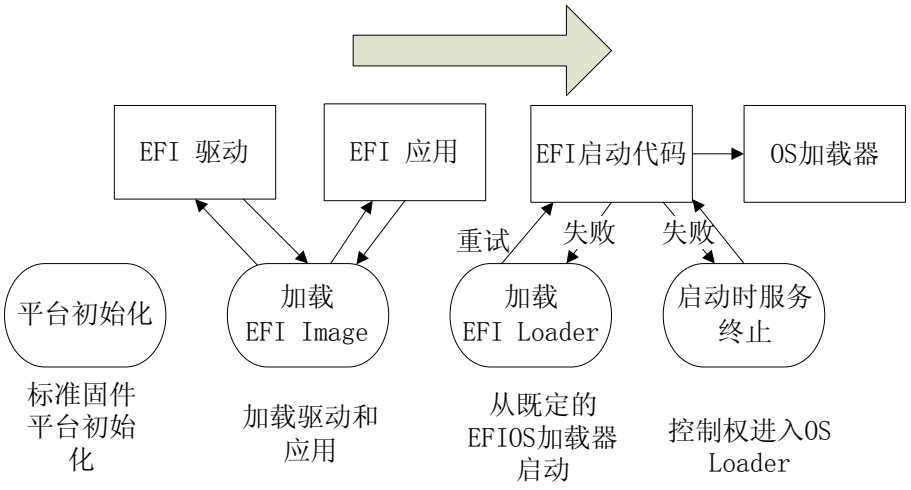

图 2.2　UEFI 的启动顺序图

UEFI 允许可扩展的平台固件通过加载 UEFI 驱动和 UEFI 应用程序映像的方式完成启动序列。当 UEFI 驱动和 UEFI 应用程序被加载的时候表明系统进入到 UEFI 运行时服务和 UEFI 启动时服务。如图 2.2 是 UEFI BIOS 的启动顺序图。

基于 UEFI 框架的平台在启动过程中，首先进行平台固件的初始化工作，完成初始化之后就开始加载 EFI 驱动程序和 EFI 应用程序。这一步是为了完成下一步初始化的功能，紧接着进行一种特殊的 EFI 应用程序的加载——操作系统加载器(OS Loader)[15]，完成了这步之后系统的控制权就由操作系统加载器掌握，此时 EFI 的启动时服务将被终止，而 EFI 运行时服务在进入操作系统后依然有效。UEFI BIOS 的整个启动过程是由启动管理器所控制。

2.1.2 UEFI BIOS 运行阶段分析

UEFI 只提供了运行于底层硬件和操作系统之间的一个平台固件规范，具体的实现由 BIOS 厂商完成。英特尔公司对这一规范做了具体的实现，发布了替代传统 BIOS 的 Tiano 项目代码。UEFI 的启动分为七个阶段进行。具体的过程如图 2.3 所示。

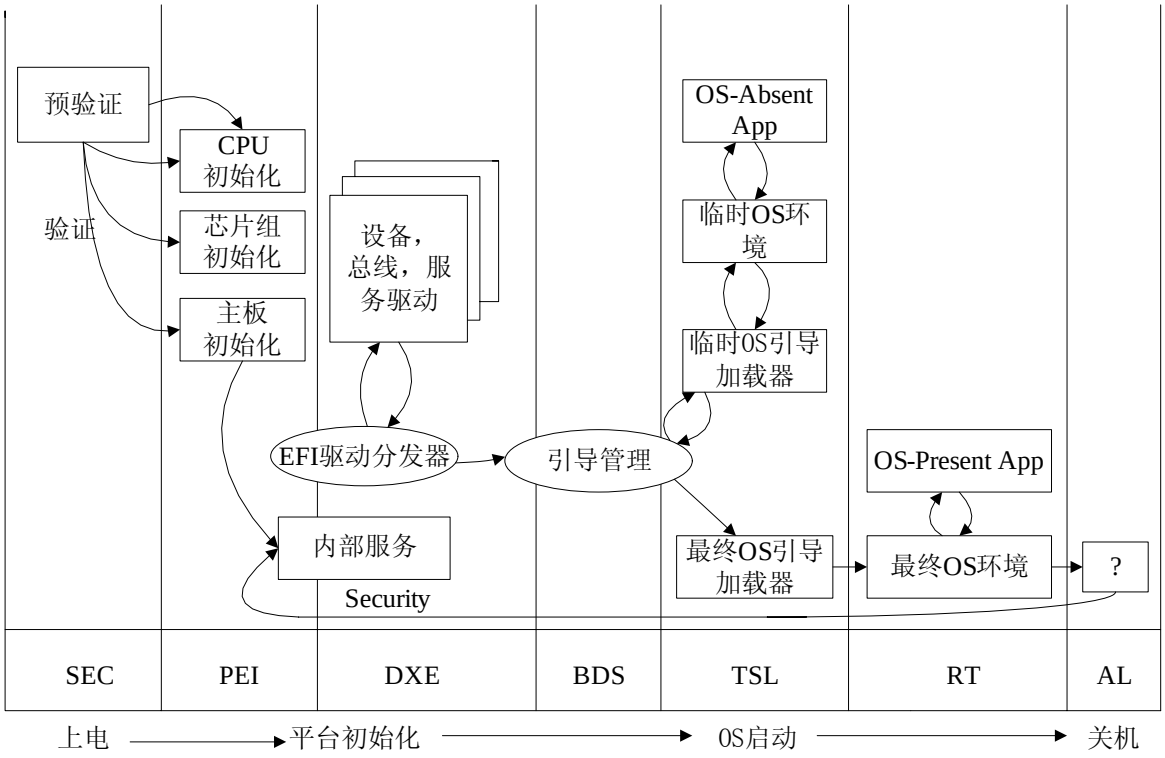

图 2.3 UEFI BIOS 的运行阶段

（1）Security(SEC) 阶段

SEC 阶段是计算机上电之后执行的第一个阶段，它是整个计算机系统可信的源头。SEC 阶段通过 CPU 使得一些临时内存产生，而这些内存可能是系统早期所能使用的临时内存。SEC 阶段只需要 CPU 预先了解平台配置。

（2）Pre EFI Initialization(PEI) 阶段

PEI 阶段是 SEC 阶段的后续阶段，主要完成 CPU、主板和芯片组的初始化工作。这个阶段的主要任务是使得系统的初始化到一定的水平，以使 DXE 阶段获得充分的运行环境。PEI 阶段起码要完成得到系统启动路径，完成最少需要的系统 RAM 和一些固件卷。这个阶段与其它的阶段相比更倚重 CPU 的结构。

（3）Driver Execution Environment(DXE) 阶段

DXE 阶段是 PEI 阶段的后续阶段。DXE 阶段的进行需要 PEI 阶段内存和资源表的准备。而 PEI 阶段代码不可重复使用的，因为 PEI 阶段到 DXE 阶段的移交是不可逆的，一旦 DXE 阶段完成载入程序，PEI 阶段代码将无效。DXE 阶段是一个具有完整设备的操作环境，为操作系统的启动提供所需的服务。

（4）Boot Device Select(BDS) 阶段

BDS 阶段是 DXE 阶段的后续阶段，称作启动设备选择阶段，它和 DXE 阶段一起实现控制台功能。用户可以从可用的启动设备中对启动项进行选择，根据选择的启动项来决定系统控制权由哪个启动设备掌握。这个阶段是系统控制权交由操作系统之前的最后一个阶段。

（5）Transient System Load(TSL) 阶段

TSL 阶段是 BDS 的后续阶段，它会依照 BDS 阶段的选择结果进行相应的操作。如果在 BDS 阶段选择了系统启动到操作系统，系统的控制权会转移到操作系统载入程序；如果在 BDS 阶段选择 EFI 运行环境，系统的控制权会移交到临时的操作系统载入程序，临时的可供 EFI 应用程序执行的环境将建立起来。

（6）Run Time(RT) 阶段

RT 阶段是 TSL 的后续阶段，该阶段操作系统获得控制权。当系统进入 RT 阶段时，UEFI 启动时服务的代码和 DXE 代码都不能被调用，只有 UEFI 运行时服务和

EFI 系统表能被正常使用。

（7）After Life(AL) 阶段

AL 阶段是最后一个阶段。这时，系统的控制权又从操作系统转移到平台固件手中。操作系统通过调用 UEFI 启动时服务或者进入 ACPI(高级配置与电源管理接口标准)的睡眠状态进入到 AL 阶段。这样做的好处就是，当系统因某个错误而崩溃之后，可以通过强制进入 AL 阶段使得计算机可以从系统或者硬件故障中恢复过来。

以上七个阶段就是 UEFI 的各个运行阶段。从以上分析可以得出：平台固件除了完成了传统的 BIOS 的基础功能之外，还实现了许多其它功能。比如对高级语言的执行环境的建立，对设备驱动的调用等。

2.2 可信计算平台

所谓平台是一种能向用户发布信息或从用户那里接收信息的实体[16]。可信计算机组对“可信”的定义是：“一个实体在实现给定目标时，如果它的行为总是符合预期，则认为该实体是可信的”。可信计算机组的这个定义，将可信任和安全区别开来。它强调的是可预测性，而不等于行为就是安全的。

可信计算平台和传统的安全保护方式(防火墙防护、病毒防护)相比，其最大的不同，就是引入了第三方受信任的平台模块 TPM 到计算机系统中，从底层硬件层面来采取高级别的防护，以密码技术为支持，以安全操作系统为核心[17,18]。通过对受信任的平台模块的度量和验证，来实现整个计算机系统的可信。

2.2.1 可信平台模块

可信计算平台的核心是可信计算平台模块(TPM)，它通常具有密码运算能力和存储能力，内部具有受保护的安全存储单元，是一个含有密码运算部件和存储部件的小型片上系统，它具有独立运行的 CPU 运算部件，作为可信平台的构件，TPM 是整个可信计算平台的基石，TPM 向操作系统、TSS 等提供诸多安全方面的支持，通过 TPM 的功能支持，可信计算平台能够提供多种安全服务。它有以下几个主要的功能：对称/非对称加密，安全存储，完整性度量和安全签名[19]。

（1） 对称/非对称加密

TPM 内部存储少量的加密数据，对称密钥存储可以将存储空间无限扩展，这样就能够任意地存储被加密的对称和非对称密钥。TPM 中最多可以存储 256 位被加密的对称密钥。这些密钥用于加密任意大小的文件。使用密钥的对称算法由开发者选择，256 位的对称密钥长度选取主要是确保能够使用 AES 算法。

TPM 内部也可以用来存储多种非对称密钥[20]。包括 512 位、1024 位和 2048 位在内的多种 RSA 密钥可以在 TPM 外部或内部产生，密钥可以在适当授权下被复制或者迁移到其他 TPM 上。如果密钥由 TPM 内部产生，并且是不可迁移的密钥，那么密钥的安全性将会更高。而可迁移密钥也可以在 TPM 的内部产生，但是它的安全性就相对要弱一点。

（2） 安全存储

在激活(或重置)TPM 时，TPM 将会利用随机数产生器产生一个新的基于 2048 位的存储根密钥 SRK[21]。该密钥的特殊性在于，它总是存在于芯片内部，并且是一个不可迁移的存储密钥。SRK 要求是不可迁移的 2048 位或者位数更长的存储密钥，其它参数的设置权限都交给终端用户。这些参数包括使用该密钥是否要求授权以及是何种授权，也包括使用该密钥是否要求提供某些 PCR 值。

（3） 完整性度量

所谓完整性度量机制，就是将完整性度量值保存在 TPM 芯片的 PCR 寄存器中，并将完整性度量时间的日志信息保存在制定的事件对象数据存储区，完整性度量值保存的是完整性度量的结果，其保存方式是将完整性度量值以扩展机制存储在 PCR 寄存器中，完整性度量事件的日志数据保存的是完整性度量的对象和度量过程等信息[22]。

平台配置寄存器(Platform Configuration Register，PCR)是 TPM 上的数据存储区。当系统第一次启动时，PCR 全部清空为零。然后，通过“扩充”的操作，即通过连接一个值和 PCR 的当前值，再完成 SHA-1 散列运算，并将产生的结果作为 PCR 当前值的替代值，外部的历史文件将会记录扩充后的 PCR 值。由于 SHA-1 散列运算是不可逆的，因此可以通过 PCR 值来校验历史文件中的值。

一个 TPM 至少有 16 个 PCR[23]。理论上说，一个 TPM 只需要一个 PCR 值来校

验历史文件中的所有值，但是 PCR 除了用于校验历史文件之外，还有一些其它的功能，所以在启动过程中会用到多个 PCR。TCG 规范中制定了早期 TPM 的平台配置寄存器的标准。如表 2.1 所示。

表 2.1　平台配置寄存器标准

PCR	用途
0	核心 BIOS、POST BIOS、嵌入可选 ROM
1	主板设置
2	可选 ROM 代码
3	ROM 配置数据
4	IPL 代码
5	IPL 配置数据
6	状态转换(待机、休眠等)
7	为 OEM 保留
8~15	未分配

装配 TPM 的计算机和只拥有可移动加密设备的计算机的不同之处就在于，TPM 能够保证机器启动时是安全的，这样就可以通过 TPM 使用平台配置寄存器(PCR)来记录主机启动的信息[24]。此外，当一个新的用户使用机器时需要重启 TPM。这个特点保证了原始用户的秘密性和安全性。

在 TPM 中，PCR 与口令的使用类似，如果 PCR 中的值与 TPM 中的值不匹配，TPM 会拒绝使用签名密钥或发布对称密钥。与使用口令的不同之处在于，这些值读取简单，而设置却相对复杂。

当平台引导完毕后，操作系统内核开始控制系统，通过读取与预期的一个或一组 PCR 值相绑定的值(或使用密钥来对一个随机数签名)来验证平台状态是否正常。如果该值可用(或该密钥能后签名)，操作系统就能得出 PCR 值处于正常状态的结论。

（4） 安全签名

可信任计算机组设计的密钥保存在硬件之中，签名则是在外部无法访问的硬件的私密存储区域进行。因此存放密钥的硬件与任意的签名都相互关联。另外，口令

会被存放在对应的口令块中，在这些数据块的头部会由不同的 2048 位公钥在不同系统中加密，并被封装。不经攻击一台单独的计算机是不可行的，而且当有人试图通过猜测授权值来实施反击时，TPM 会自行通知操作系统。TPM 可以产生无限的密钥以供使用，用户可以使用不同的密钥对应签名。每个签名可能会有不同的角色和功能，可以用不同的密钥来登录不同的网站。

2.2.2 可信计算平台的组成

可信根和信任链是可信计算平台的重要部分。可信计算的基本思想是：首先建立一个可信根，由可信根开始建立一条信任链，从可信根到可信硬件平台，再到操作系统、数据库系统，最后到应用程序，一级验证一级，一级信任一级，从而把这种信任关系传递到整个计算机系统中[25]，从而保证整个计算机体系的可信性。如图 2.4 为可信平台模块的组成图。

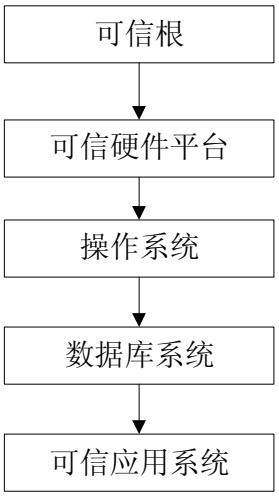

图 2.4 可信计算机系统组成

由图 2.4 可以看出，可信计算平台是结合了可信硬件和可信软件包括可信操作系统在内的一个计算机系统[26]，它是软硬件的综合体。可信计算平台重要的基础部件包括可信平台模块 TPM 和 TCG 软件协议栈 TSS(TCG Software Stack)。可信软件协议栈处于 TPM 之上，应用软件之下，是对可信计算平台提供支持的软件，它的设计目标是对使用 TPM 功能的应该程序提供一个唯一入口，并提供对 TPM 的同步访问管理。TSS 平台软件从结构上可以分为四层，自下至上分别是 TPM 驱

动程序库(TPM Device Driver Library, TDDL)、可信软件栈核心服务(TSS Core Services, TCS)和可信服务提供者(Trusted Service Provider, TSP)[27]，TSS 的体系结构如图 2.5 所示。

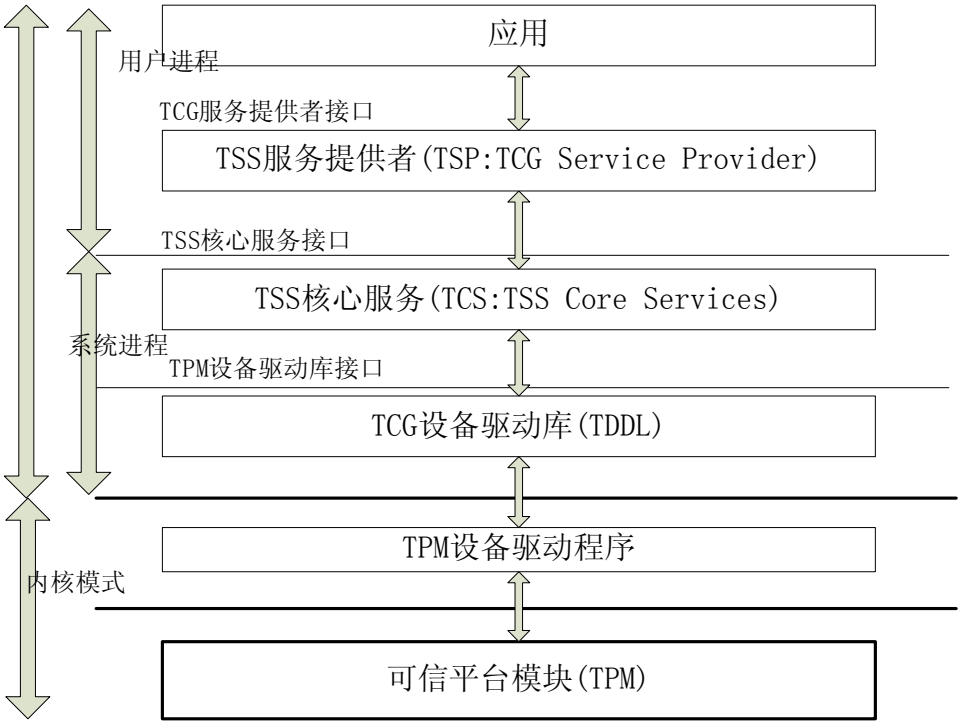

图 2.5　TSS 体系结构图

TDDL 是一个提供与 TPM 设备驱动程序进行交互的 API 库[28]。一般来说，TPM 生产厂商会随 TPM 设备驱动程序一起附带 TDDL 库，以方便 TSS 实现和 TPM 进行交互。TDDL 提供一个小的 API 集合来打开和关闭设备驱动程序，发送和接收数据块，查询设备驱动程序的属性，以及取消已经提交的 TPM 命令。

TCS 层有以下几个任务：管理 TPM 的资源，比如授权会话和密钥上下文的交换；提供一个 TPM 命令数据块产生器，这个命令数据块产生器能把 TCS API 请求转换成 TPM 能后识别的字节流；提供一个全局密钥存储设备，并同步来自 TSP 层的应用程序访问[29]。

TSP 层以共享对象或动态链接库的方式直接被应用程序调用。TSP 接口(Tspi)对外提供了 TPM 的所有功能和它自身的一些功能[30]，比如密钥存储和弹出关于授权数据的对话框。

2.3 BitLocker 驱动器

BitLocker 驱动器是 Windows Server 2008 及以上操作系统下可选的数据保护功能。其功能的最终实现依赖于固件对 TCG BIOS 的支持和主板上集成了 TPM 芯片。

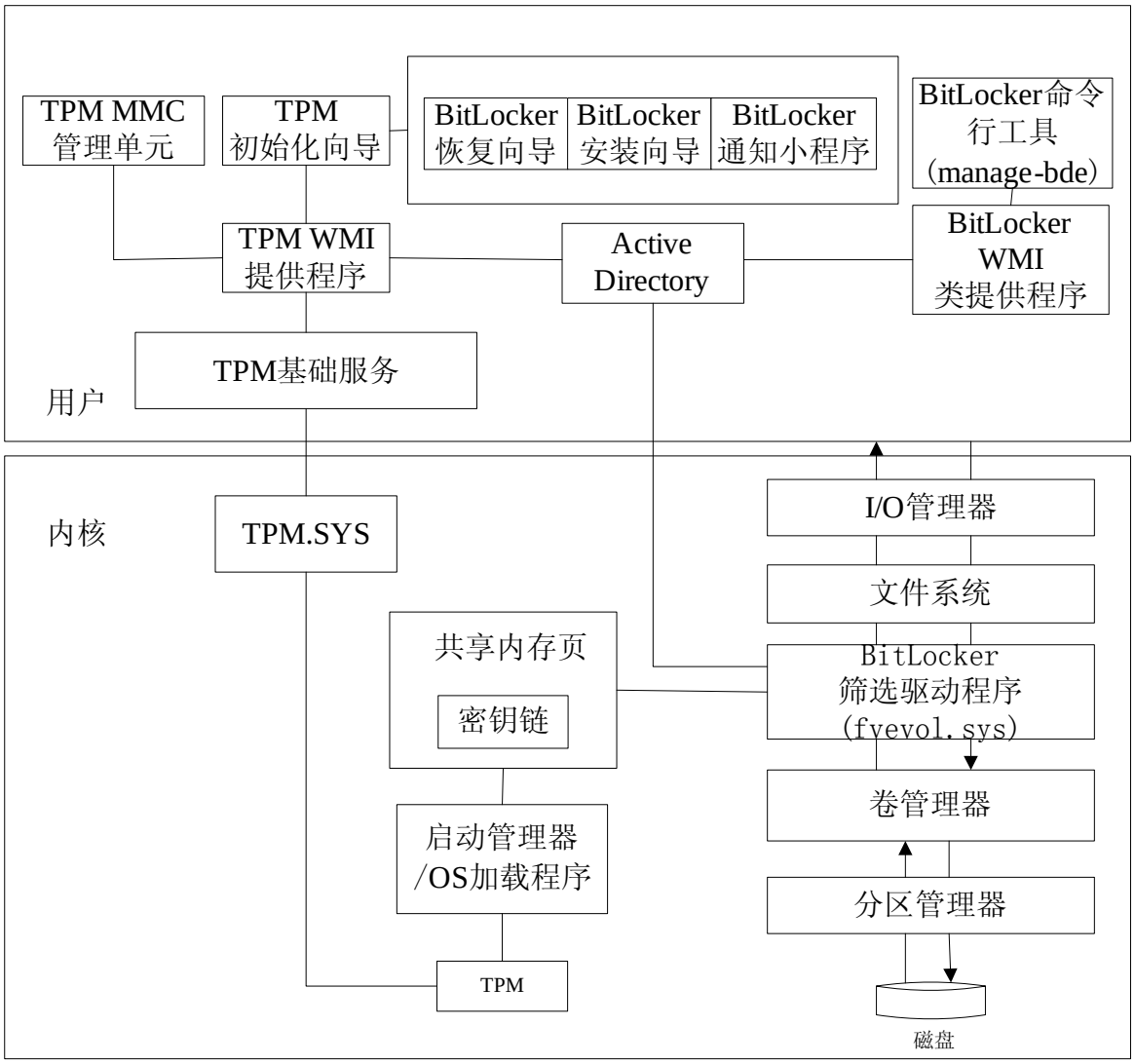

图 2.6 BitLocker 的体系结构

图 2.6 显示了整个 BitLocker 的体系结构，包括它的各种子组件。它显示了 BitLocker 的用户模式和内核模式组件，包括 TPM 以及它们与不同层的操作系统集成的方式。

BitLocke 有助于在计算机脱机时防止对硬盘的操作系统卷进行未经授权的访问。为了实现该功能，BitLocker 使用全卷加密和 TPM 提供的安全增强。在具有 TPM 的计算机上，BitLocker 还支持多重身份验证。

BitLocker 使用 TPM 对早期启动的关键组件执行系统完整性检查。TPM 收集和存储多个早期启动组件以及启动配置数据的度量以为该计算机创建系统标识符，就像指纹。如果早期启动组件发生更改或被篡改，如更改 BIOS、更改主启动记录(MBR)或将硬盘移动到另一台计算机，则 TPM 防止 BitLocker 对加密卷和进入恢复模式的计算机进行解锁。如果 TPM 验证系统完整性，则 BitLocker 对受保护的卷进行解锁。然后启动操作系统，系统保护便成为用户和操作系统的职责。

BitLocker 对受保护的操作系统卷的访问进行身份验证之后，当将数据写入受保护的卷或从受保护的卷中读取数据时，Windows Vista 文件系统堆栈中的筛选器驱动程序对磁盘扇区透明地进行加密和解密。当计算机休眠时，休眠文件以加密形式保存到受保护的卷。当计算机从休眠中恢复时，对加密的休眠文件进行解密。在设置期间 BitLocker 加密受保护的卷，通常加密和解密对日常系统性能的影响非常微小。

如果临时禁用 BitLocker(例如，要更新 BIOS)，则操作系统卷仍然保持加密，但会以未加密形式存储在硬盘上的“明文密钥”对卷主密钥进行加密。使用此未加密的密钥会禁用 BitLocker 提供的数据保护。当重新启用 BitLocker 时，将从磁盘中删除未加密的密钥，对卷主密钥再次执行设置密钥和加密操作，BitLocker 保护恢复。

IT 管理员可以通过 BitLocker 设置向导本地配置 BitLocker，或借助于 Windows Vista 操作系统的 Win32_EncryptableVolume WMI 提供程序提供的界面进行本地和远程配置。界面包括对卷开始、暂停和恢复加密以及配置如何保护卷的管理功能。

2.4 本章小结

本章首先介绍了 UEFI BIOS 与传统 BIOS 相比的优点，然后又简要介绍了 UEFI BIOS 的框架结构和运行阶段分析。在可信计算平台方面，首先介绍了可信平台的核心模块——可信平台模块，在对它的功能有了一定了解的基础后，介绍了基于可信计算思想下的可信计算平台的硬件组成和体系结构。

3 可信链的设计

可信计算平台的可信性是在启动过程中，由平台的可信根出发，通过信任的扩展机制在可信计算平台各硬件部件、软件部件间传递信任形成的，而信任扩展机制的实现要依赖完整性度量机制，由完整性度量机制实现可信性的传递，形成信任链，并最终保证整个平台的可信。

可信根在可信计算平台中分别为可信度量根(Root of Trust Measurement)，可信存储根(Root of Trust for Storage，RTS)和可信报告根(Root of Trust for Reporting)。RTM 是负责可信度量值的计算任务的，它是可信平台信任度量的基石。RTS 是用来存储主密钥的，是可信计算平台内进行可信存储的基础。RTR 是可信计算平台内可信报告的基础。可信根式整个平台可信的基础

3.1 确立可信根

可信任计算机组采用一种链式的设计来判断平台的可信状态。系统从一个很小的信任根开始启动，信任根是系统中第一个获得控制权的模块。这个信任根是 BIOS 的一部分，这个信任根也可以称作核心可信度量根(Core Root of Trust for Measurement，CRTM)[31]。在理想的情况下，核心可信度量根存储在 TPM 中，但是一般的情况下，CRTM 也可以存储在系统的 ROM 区。CRTM 记录了在将主板控制权权交给整个 BIOS 之前哪个执行过程将用来启动系统。接下来，由 BIOS 记录哪一个板卡的 BIOS 将获得控制权，然后引导加载程序记录哪一个内核将获得系统控制权，最终，内核将记录哪一个操作系统引导之后哪些程序将会获得系统的控制权。

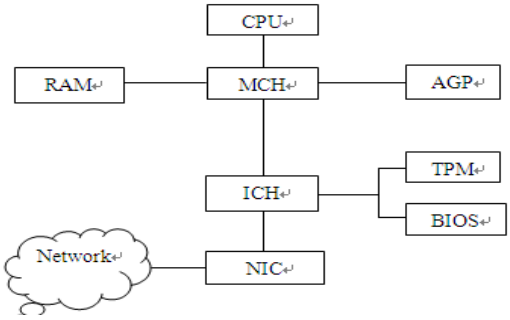

图 3.1 TPM 在系统中的位置

由图 3.1 可以看出可信计算平台的硬件结构，可信平台模块 TPM 芯片选择了和 BIOS ROM 芯片一样的低速接口，连接在南桥上。这样做的原因是出于以下两方面的考虑：一是 TPM 芯片自身作为一个低速密码处理芯片，不需要高速接口；二是因为建立在可信平台信任链的起始点，需要硬件平台最初的软件代码，在这里就是平台运行最初的执行代码 BIOS。TPM 芯片需要同 BIOS 协同运行，以便建立可信根，将它们放在同一硬件总线上，便于在系统加电之后，不需要初始化过多的设备就可以进行可信根的建立。

3.2 基于 UEFI 的可信链设计

由于 BIOS 协同 TPM 芯片作为整个系统上电之后的源头，所以需要依赖于 UEFI BIOS 的各个运行阶段，才能完成整个系统启动过程的度量。

根据第二章中关于 UEFI BIOS 的运行阶段分析，可以知道 UEFI BIOS 启动分为了七个阶段：Security(SEC)阶段、Pre-EFI Initialization(PEI)阶段、Driver Execution Environment(DXE)阶段、Boot Device Select(BDS)阶段、Transient System Load(TSL)阶段、Run Time(RT)阶段、After Life(AL)阶段。

终端设备上电之后，SEC 阶段第一个执行，会进行上电自检并进行硬件信息的验证。从总线周期意义上来说，当使用 CPU 缓存在非易失性工作模式下，不会导致芯片组对该区域进行以外的回写操作。SEC 阶段执行完毕进入到 PEI 阶段，该阶段仅进行 CPU、芯片组和主板的初始化工作，执行代码相对较少。PEI 阶段执行完毕之后，进入 DXE 阶段。这三个阶段的代码都是存储于系统的 ROM 中，只有当系统硬件上有所改动(如设置跳线)时，这三个阶段才会有所变化。因此，无法通过软件来改动这三个阶段。所以，可以认为 SEC、PEI、DXE 三个阶段都是建立在硬件防护水平的，可以把 UEFI BIOS 的这三个阶段作为后续执行阶段的度量根。

有了度量可信根，就可以对 UEFI BIOS 接下去的阶段度量。在进入 DXE 阶段之后，首先扫描 Option ROM，获取需要加载的驱动文件和应用程序列表，之后进行完整性度量，如果验证通过则允许加载，若没有验证通过就需要停止加载，并报告管理员，更新事件日志。在 TSL 阶段和 RT 阶段也有同样的验证过程，分别对操作系

统加载器(OS Loader)、操作系统核心代码(OS Code)进行验证，通过 TPM 完成 PCR 值和事件日志的更新。当系统的控制权由操作系统掌握后，需要对操作系统的各种应用程序进行验证，信任关系依次传递下去，直到完成操作系统及其应用程序的加载，这样整个系统的信任链就建立起来了。如图 3.2 所示，为可信链的建立过程。

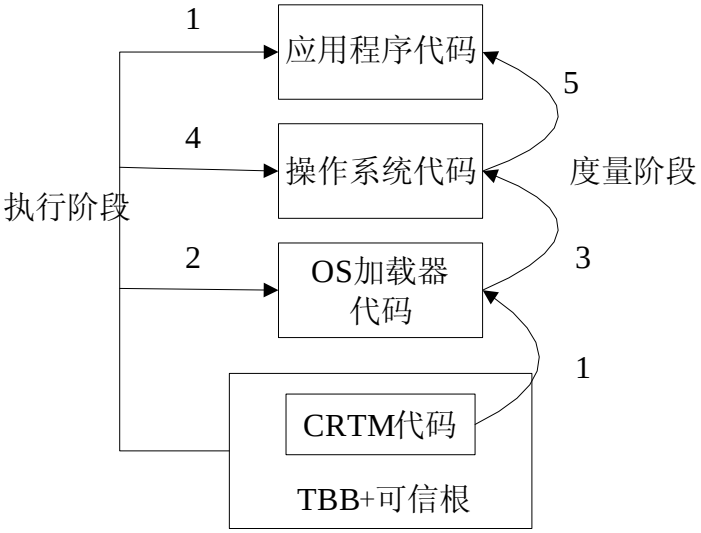

图 3.2　建立可信链的过程

如图 3.2 中，1、3、5 代表了度量过程，2、4、6 步骤代表了系统正常的启动序列。TPM 在这个过程中并不会去禁止引导一个不安全操作系统或使用一个不安全的引导加载程序，而是仅仅记录系统的启动序列，对这些序列的验证结果用于决定系统启动序列是否交由下一个可信的部件执行。系统就通过这样一种模式得到可信的启动。

3.3　平台完整性度量机制

要保证可信平台开始工作后运行在一种可预知的状态，就需要保证系统的完整性没有遭到破坏，保证系统没有被病毒或木马程序修改。TPM 设备可以提供可信报告平台完整性的任务。完整性度量机制主要包括完整性度量计算、完整性度量存储、完整性度量报告和完整性度量验证四个部分。

完整性度量计算是指对平台完整性相关的特征进行度量，并把度量结果的摘要保存在平台寄存器 PCR 中的过程。

完整性度量存储是完整性度量和完整性报告的一个中间步骤。存储对象包括完整性度量事件的日志信息和在 TPM 内部的 PCR 中的存储完整性度量摘要值，其中完整性度量摘要值的存储基于 PCR，它是 TPM 的重要组成部件。

完整性度量报告是指提供完整性度量存储的内容，其关键就是能够提供可信的 PCR 值和可信的完整性度量事件日志信息。

完整性度量验证是对完整性度量报告的数据，按度量的过程执行验证计算，得出当前完整性度量报告所反映的平台状态可信与否的结论。

3.3.1 完整性度量计算

完整性度量计算在 TPM 的内部完成。通过 TPM 的 SHA-1 引擎哈希计算摘要值。SHA-1 计算在 TPM 内部通过会话的方式实现，其实现流程如图 3.3 所示。

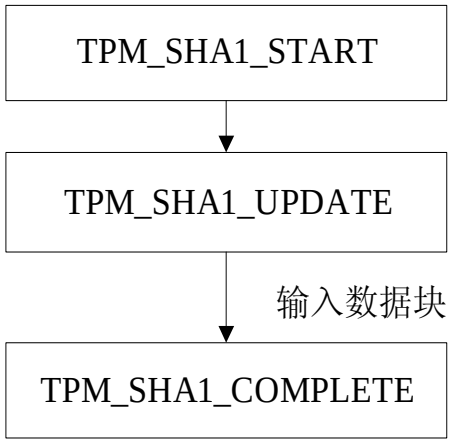

图 3.3 度量计算的实现流程

SHA-1 的计算完全在 TPM 的内部完成。其调用的函数都属于密码学功能型函数。首先调用 TPM_SHA1_START 接口函数开始一个计算 SHA-1 摘要的过程，接着调用 TPM_SHA1_UPDATE 函数，输入完整的数据块。最后调用 TPM_SHA1_COMPLETE 函数完成 SHA-1 计算，并将结果返回。

3.3.2 完整性度量存储

完整性度量存储主要就是将完整性度量结果存储到平台配置寄存器 PCR 中，并将完整性您该度量日志的信息保存到指定的事件对象存储区。其实现流程如图 3.4 所示。

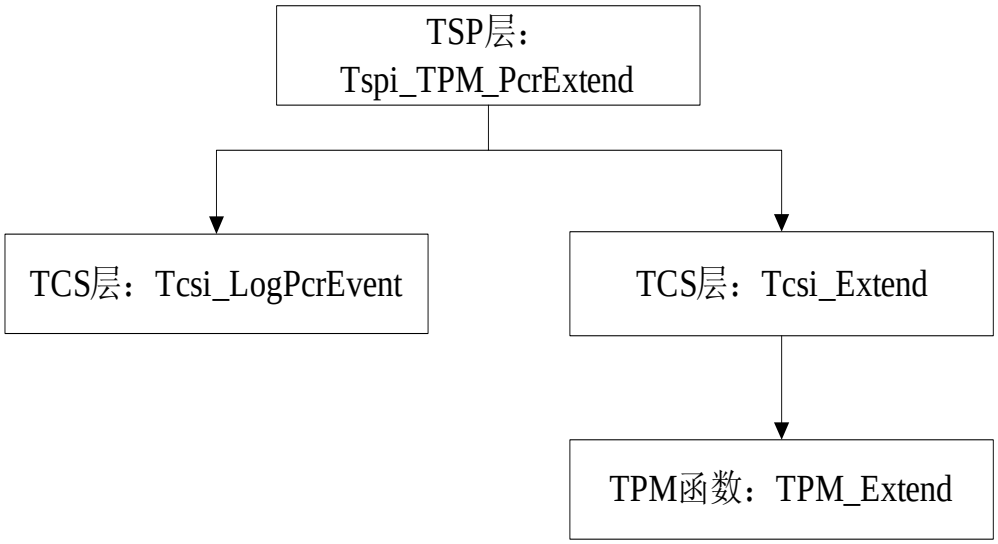

图 3.4　完整性度量存储实现流程

完整性度量存储流程首先通过调用 TSP 层的 Tspi_TPM_PcrExtend 函数实现以当前完整性度量值将某个 PCR 的值扩展更新，并将其保存在对应的 PCR 中。Tspi_TPM_PcrExtend 函数向下调用 TCS 层函数，Tcsi_LogPcrEvent 函数用以实现在制定 PCR 对应的度量 日志数据存储区的末尾区域保存当前完整性度量事件的日志信息。Tcsip_Extend 函数用以引起对某个 PCR 值的更改。Tcsip_Extend 函数再调用 TPM_Extend 函数，最后由该函数负责将一个新的完整性度量值以扩展机制保存入一个指定的 PCR 中，完成度量值的存储。

3.3.3　完整性度量报告

完整性度量报告是指平台向验证实体提供平台或部分部件的完整性度量值，并向验证实体提供完整性度量相关信息，以使验证实体能够通过完整性验证，判断平台的当前状态。此外，完整性度量报告还应该包括一个验证过程。

完整性度量信息的获取，主要包括完整性度量值的获取和完整性度量日志的获取两个部分。

（1）完整性度量值的获取实现

完整性度量值保存在 TPM 的平台配置寄存器中，通过读取寄存器实现值获取。读取的流程如图 3.5 所示。

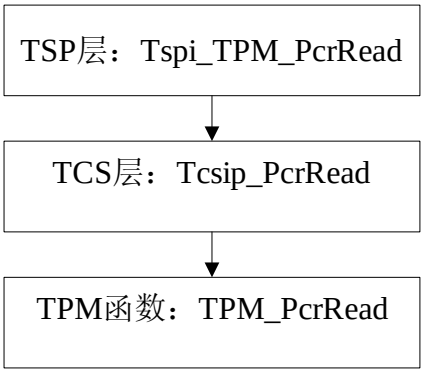

图 3.5　读取 PCR 实现

Tspi_TPM_PcrRead 函数属于 TSP 层，用于读取一个指定的 PCR 当前存储的值，然后 Tspi_TPM_PcrRead 向下调用 Tcsip_PcrRead 函数，Tcsip_PcrRead 用于提供某个指定 PCR 存储值的非加密报告，它再调用 TPM_PcrRead 函数，直接访问某个特定 PCR 并读取出该 PCR 非加密的当前值。

（2）完整性度量日志的实现

获取完整性度量日志的实现分为获取某个 PCR 的单个事件日志，获取某个 PCR 的多个连续事件日志和获取全部事件日志三种。其实现流程如图 3.6 所示。

获取单个事件日志时，调用 Tspi_TPM_GetEvent 需要指定 PCR 的索引号和所求事件的索引号，接着调用 Tcsi_GetPcrEvent 事件管理接口函数。获取某个指定 PCR 的多个连续的完整性度量事件日志，在调用 Tspi 层函数时需指定某个 PCR 的索引号和事件个数。获取全部事件日志时，调用 TSP 层函数只需要 TPM 对象的句柄即可。

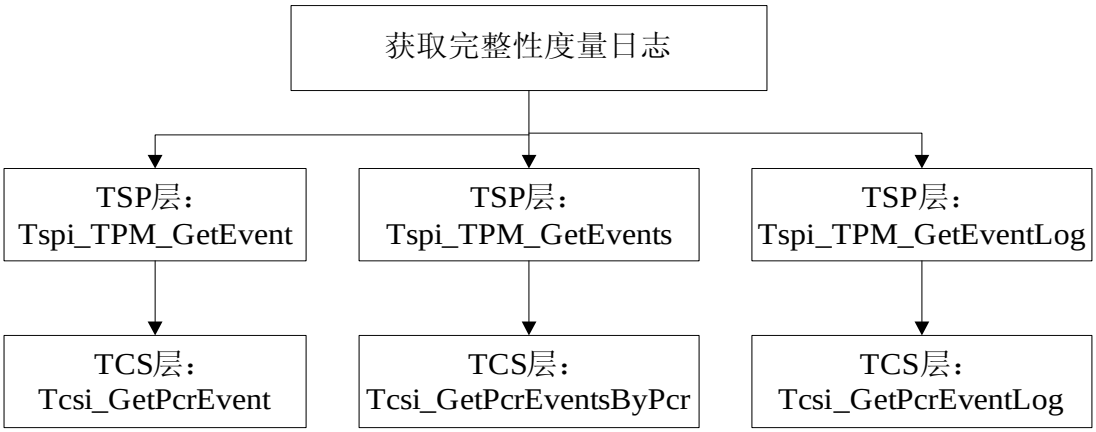

图 3.6　完整性度量日志实现

3.3.4 系统启动过程度量机制

从系统上电开始开始度量，首先从核心可信度量根开始对 BIOS 进行度量，在将控制权移交给 BIOS 之前，会将度量结果放在可信平台模块相应的 PCR 中，保存度量日志信息[32]，按照与基准值的比较判断度量结果，若得出结果 BIOS 是可信的，则核心度量可信根会把控制权交给 BIOS；接下来由 BIOS 开始对引导加载程序进行度量，将度量结果存在可信平台模块对应的 PCR 中，并保存度量日志信息，基于基准值的度量结果，若引导加载程序所对应的 PCR 记录是可信的，则信任引导加载程序，控制权移交到引导加载程序之上；接着再依次度量操作系统核心代码和操作系统的可信性，一直到将度量过程进行到应用程序，一级一级的确保可信性。这样就形成了可信平台信任关系的传递，实现了平台完整性的度量机制。如图 3.7 所示为平台完整性传递图。

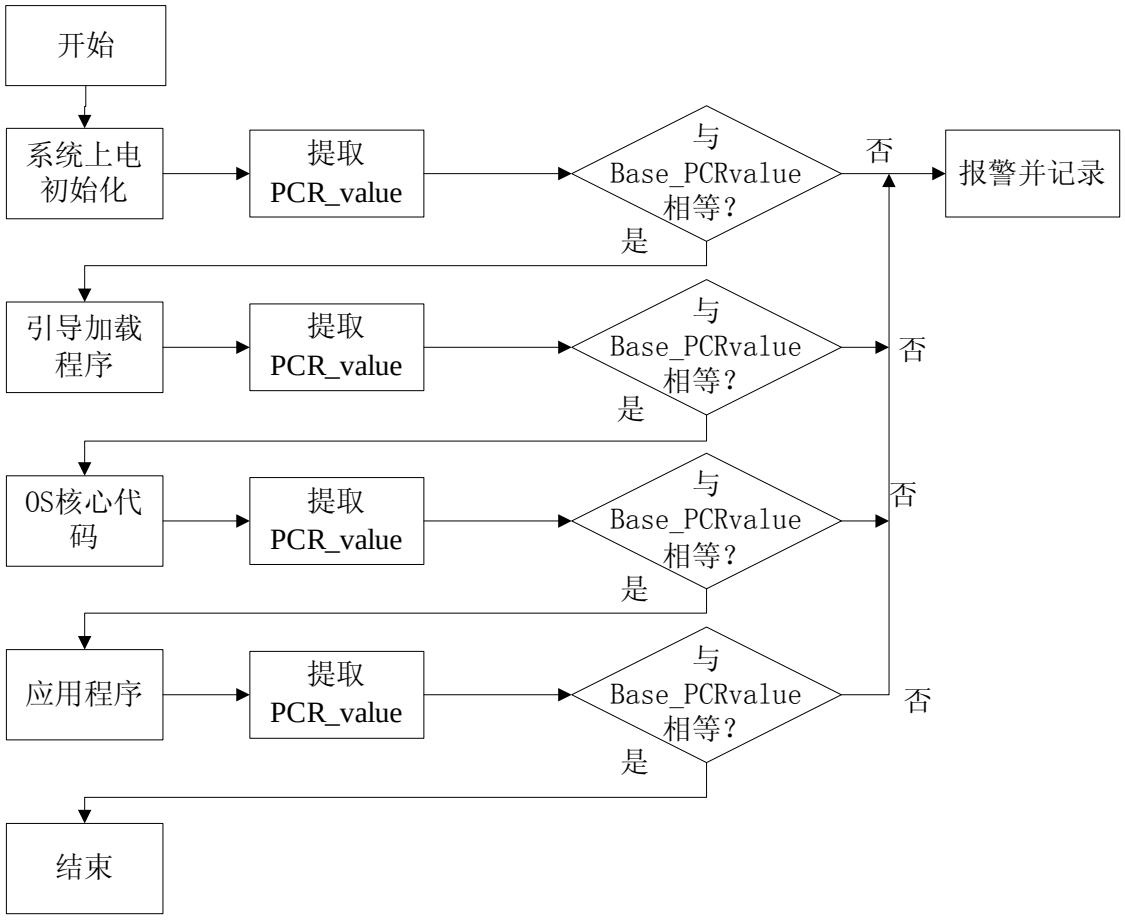

图 3.7 平台完整性传递图

3.4 本章小结

本章引入可信计算机系统的核心思想到系统的启动过程中，根据可信计算平台的硬件结构并结合 TPM 芯片的特性，将 TPM 协同 BIOS 作为整个可信链的可信度量根。根据 UEFI 的启动阶段分析，建立了基于 UEFI 的可信链，并最终借助 TPM 芯片的平台配置寄存器(PCR)实现平台完整性度量机制。

4 TPM 设备驱动程序设计与实现

本文设计的方案，其安全模块——可信任平台模块(TPM)是集成在主板上。可信任计算机组在 2006 年就制定了基于 UEFI 的 TCG 接口协议。在 UEFI 2.0 的接口标准中提供了与安全相关的接口规范：安全启动(Security Boot)、驱动签名(Driver Signing)和 Hash 算法[33]。这为该方案的实现提供了便利的条件。

可信平台模块本质上是一个以硬件方式连接到主板上的被动存储设备。TPM 位于 LPC(低管脚数)总线下，LPC 总线也用于连接系统 BIOS，这就确保了系统早期引导时，在其他设备尚未初始化之前 TPM 设备是能够使用的。TCG 设备驱动程序库(TDDL)是用来与 TPM 进行通信的[34]，TCG 软件栈中规范了它的接口。TDDL 一般与操作系统内核中的 TPM 设备驱动程序交互，TPM 设备驱动程序最终与 TPM 设备通信。其交互方式如图 4.1 所示。本章主要实现 TPM 设备驱动程序的设计。

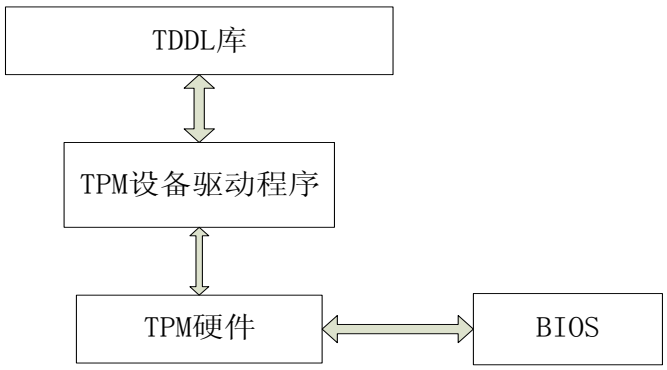

图 4.1 TPM 的通信方式

TPM1.1 标准时可信任计算机组直制定 TPM 标准，并被大部分的厂商接受和认可。然而，该标准并没有定义与实际 TPM 设备进行通信的机制。因此，每个厂商必须定义自己的底层通信接口。为了减少 TPM 设备接口和设备驱动程序的数量，可信任计算机组决定将设备接口标准化添加进 TPM1.2 规范之中。新标准的发布，使得系统 BIOS 厂商和操作系统厂商只需要实现一个驱动程序就可以支持所有兼容 TIS(TPM Interface Specification)的设备。

TIS 定义了两种不同的设备接口：一种是传统的 I/O 端口的接口方式，另一种是内存映射的接口方式[35]。

4.1 I/O 端口通信

采用 I/O 端口的方式与 TPM 芯片进行通信时，首先需要知道所使用的 TPM 设备 I/O 的端口地址。以 Atmel 1.1b TPM 芯片为例，它的 I/O 端口地址分别有 0x4E、0x4F、0x400、0x401。其中，端口 0x4E、0x4F 是用来查询 TPM 芯片的版本号和厂商信息的，端口 0x400 是数据端口，端口 0x401 是状态端口。

为了便于理解，在代码中使用如下宏定义：

```
#define TPM_DATA_PORT          0x400   /*可编程输入输出数据端口*/
#define TPM_STATUS_PORT    0x401   /*可编程输入输出状态端口*/

#define TPM_STATUS_ABORT  0x01    /*中止命令(写方式)*/
#define TPM_STATUS_BUSY    0x01    /*设备忙(读方式)*/
#define TPM_STATUS_DATA_AVAIL    0x02    /*数据准备好(读方式)*/
```

以下代码实现读取指定端口索引的值到数据端口。

```
int rdx(int index)
{
    outb(index, TPM_ADDR);                  /*读取指定端口索引的值*/
    return inb(TPM_DATA_PORT) & 0xFF;   /*将值放进数据端口*/
}
```

TPM_ADDR 代表端口地址，TPM_DATA_PORT 代表数据端口地址。

当对 0x401 端口进行读操作时，该端口对应的就是 TPM 的状态寄存器。TPM 状态寄存器的第一位代表 TPM 当前是处于忙碌还是空闲状态[36]。第二位代表数据是否准备就绪。如果把 TPM 状态寄存器的第一位置 1，将会使 TPM 终止当前正在执行的命令。端口 0x400 是数据端口，它是用来向 TPM 发送命令或者接收 TPM 所返回的值的。

对于不支持中断的 TPM 芯片，它的设备驱动程序的流程如图 4.2 所示。

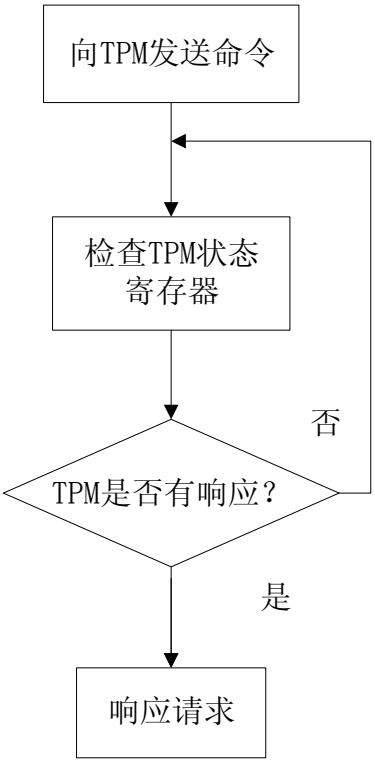

图 4.2　设备驱动程序的典型流程

图 4.2 显示了一个不支持中断的 TPM 的设备驱动程序的典型流程。首先向 TPM 发送一条命令，设备驱动程序要通过轮询的方式来判断 TPM 状态寄存器的值，检查 TPM 是否已经发出应答。当状态寄存器显示有请求时，TPM 设备才去响应请求；若状态寄存器显示无请求，驱动程序将会继续轮询检查。

4.1.1　芯片初始化

以 Atmel 的 TPM 芯片为例，由于它并不需要很特殊的初始化来保障正常的运行。TPM 芯片所挂载的 LPC 总线已经被 BIOS 初始化并进行了相关的配置。以下初始化程序，用来检查 Atmel 的 TPM 芯片是否存在。

```
int init(void)
{
    if(rdx(4)!='A'||rdx(5)!='T'||rdx(6)!='M'||rdx(7)!='L')
        return 0;                                    /*初始化失败*/
    return 1;                                        /*初始化成功*/
}
```

4.1.2 向 TPM 设备发送命令

函数 int send(unsigned char * buf , int count)实现向 TPM 设备发送命令。这个函数的首先是要确保 TPM 设备在接受命令时是处于已知状态的，要达到这个要求就需要终止当前任何正在执行的命令，然后等待设备准备就绪。

函数 Wait_For_Status 是用来等待 TPM 状态寄存器的忙碌/空闲标志位发生变化的。利用条件(Status & Mask) == Value 来检查状态寄存器的变化，如果结果为假，在短暂的等待之后会再次判断条件是否成立。在一段时间之内，如果判断条件仍然为假的话就返回 0。如果最终条件得到满足，函数会返回 1。向 TPM 设备发送命令的流程如图 4.3 所示。

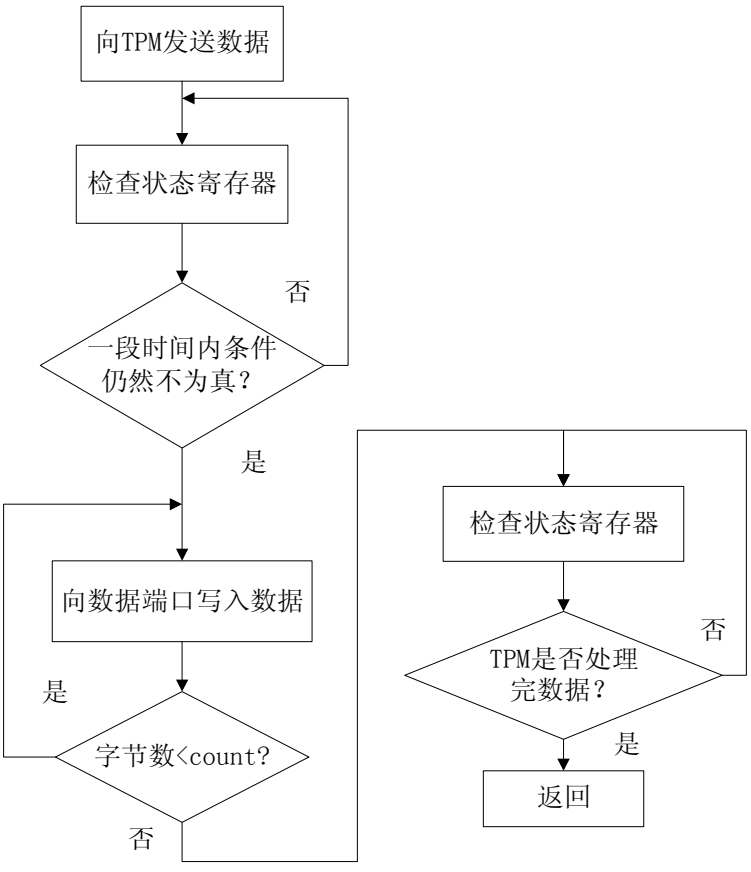

图 4.3 向 TPM 发送数据的流程图

重要代码实现如下：

```
for (i = 0; i < count; i++)
    outb(buf[i], TPM_DATA_PORT);        /*向端口写入 n 字节数据*/
```

4.1.3 TPM 设备接收数据

当 TPM 设备准备就绪时，就会向数据端口 0x400 一次写一个字节，写入的内容应该遵循可信任计算机组对 TPM 规范定义的命令字节流。函数 recv(unsigned char * buf, int count)用来实现这个功能。当命令字节流写完后，需要首先检查 TPM 设备是否正在处理设备以及设备是否处理完毕并返回结果。函数 Wait_For_Not_Status 用来检查这两种情况，该函数测试的不等式为：(Status & mask) ! = value。这个函数用来检查忙碌/空闲标志位和可用标志位。

在调用接收数据的函数 recv 之前，可以用过检查状态寄存器的忙碌/空闲标志位来判断设备当前是否处于空闲状态，如果设备在一段时间内一直忙碌的话，可以选择调度另一个任务。这是因为如果系统一直检查忙碌/空闲标志位的话，会使系统运行比较缓慢。TPM 接收数据的流程如图 4.4 所示。

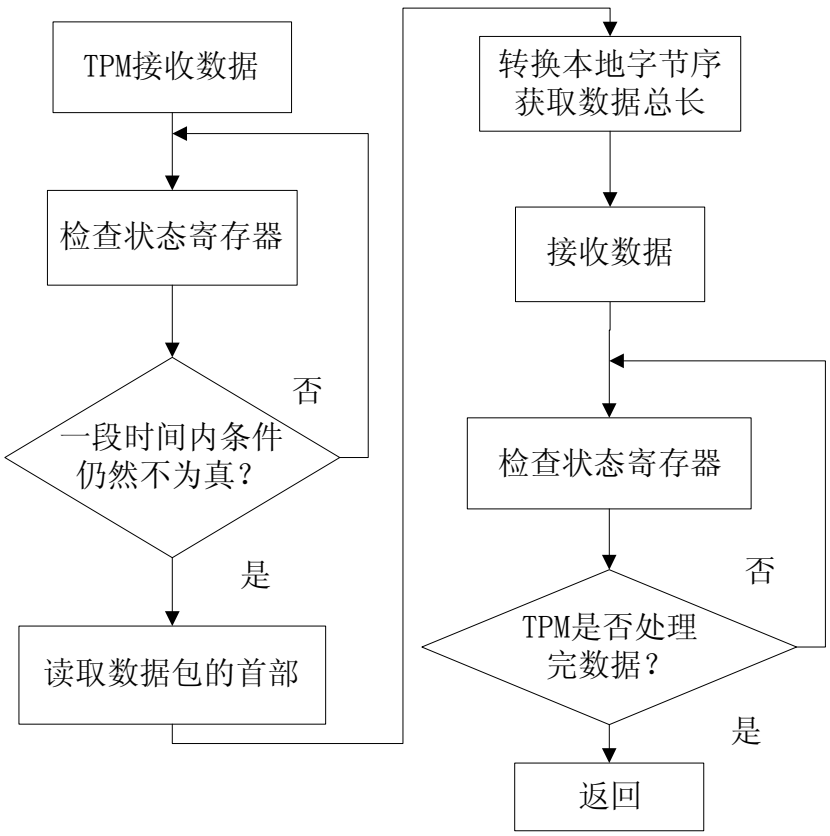

图 4.4 TPM 接收数据的流程图

读取数据包首部实现

```
for (i = 0; i < 6; i++)
{
    if (inb(TPM_STATUS_PORT) & TPM_STATUS_DATA_AVAIL == 0)
        return 0;
    *buf++ = inb(TPM_DATA_PORT);
}
```

4.2 内存访问方式通信

在支持 TPM1.2 规范的 TPM 设备，可以用内存对 I/O 端口进行映射[37]，TPM 芯片预留了一部分物理内存空间，使得主机的操作系统可以将这部分内存空间映射到相应的虚拟地址空间。

4.2.1 技术细节

TPM 的内存映射方式如图 4.5 所示。它由 5 个 4KB 大小的页组成，每个页都表示一类相似的寄存器组。每个页面对应一个 Locality，也就是说，通过位于地址 FED40000 的寄存器组发送到 TPM 的命令和 Locality 0 有关。同样，通过位于地址 FED42000 的寄存器组发送的命令和 Locality 2 有关。TPM 的寄存器组有多个副本并采用页边界对齐的方式，这些是为了使主机操作系统可以为不同级别的系统分配不同的 Locality，并通过控制这些系统的虚拟内存映射来限制它们对通道的访问。一个安全的内核只允许访问 Locality 1 上的寄存器组，而应用程序则可以操作 Locality 3 上的寄存器组。Locality 0 上的寄存器组是为了兼容以前的 1.1b 规范的。Locality 4 被处理器用作存储安全加载程序的度量值，这个值存放在 PCR17(TPM1.2 规范扩展了 PCR16~23)中，这一过程只能通过特殊的 LPC 总线周期处理，任何软件都无法产生这个总线周期，这就保证了安全加载程序的度量值是通过处理器产生的。

在内存映射访问模式下，可能会有多个用户在同一时刻访问 TPM 设备，这就会引发同步的问题。例如，一个可信的操作系统和一个应用程序可能同时给 TPM 送请求，而为了确保 TPM 内部状态的一致性，TIS 开发了一个锁协议，在用户通过某个 Locality 使用 TPM 之前，需要先将访问寄存器的访问请求位(RequestUse)置 1，如果

当前 TPM 没有被使用的话，那么就允许 TPM 接受访问；如果当前 TPM 设备正在被某个 Locality 使用，那么访问请求将被挂起，直到当前的 Locality 不再使用 TPM。最高界别的 Locality 将最先获得 TPM 设备的访问权，如果一个 Locality 出现问题或者正在被恶意使用而没有交出控制权，那么较高级别的 Locality 将通过设置 Seize 位来获得 TPM 的控制权。

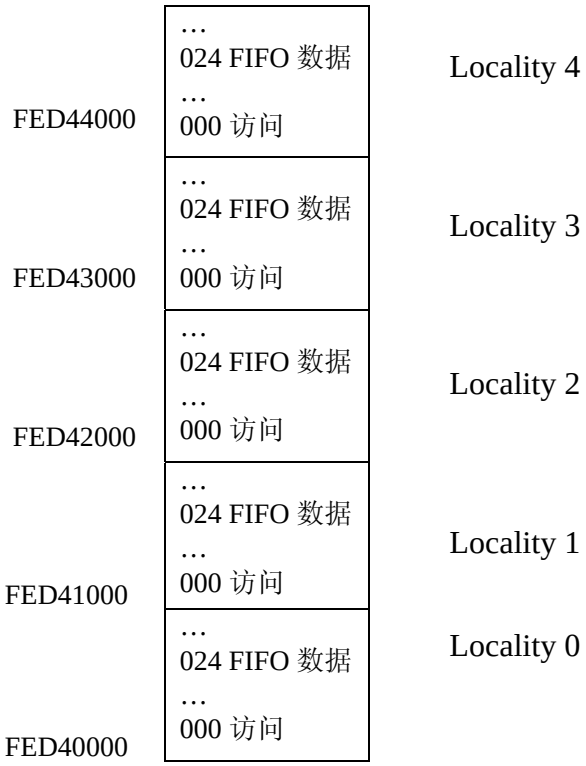

图 4.5　TPM 内存映射图

TPM 提供的寄存器包括访问寄存器、状态寄存器和 FIFO(先入先出)数据寄存器。这些寄存器分别处在偏移量 000h、018h、024h[38]。当写访问寄存器时，向第二位写一个 1 表示当前 Locality 发出一个使用 TPM 的请求，向第六位写一个 1 表示当前 Locality 结束使用 TPM。当读访问寄存器时，第六位表示当前 Locality 正在使用 TPM。

状态寄存器是一个 8 位(A_0~A_7)的寄存器[39]，它的每一位分别代表：

A_0：标识 TPM 当前是否处于忙碌状态；

A_1：标识数据是否已经准备好；

A_3：标识软件要求 TPM 继续向数据寄存器中发送数据；

A_4：标识 TPM 发出了相应；

A_5：标识 TPM 正在准备接收命令；

A_7：标识 TPM 的状态是有效的；

A_2 和 A_6 作其它使用。

一些寄存器组(如 Locality 0)包含了额外的寄存器。如，偏移量 0x0F00 处的设备寄存器和厂商 ID 寄存器，该寄存器一般用于确定 TPM 的型号和厂商的型号。

4.2.2 初始化设计

在使用 TPM 之前，首先要初始化 TPM 设备。初始化函数 int()首先强制关闭所有的 Locality，然后请求使用 Locality 0。当允许访问后，将检查是否存在有效的厂商 ID，如果检查通过，才能对 TPM 设备进行其他操作。

为了便于理解，在代码中使用下面的宏定义：

```
/*Locality 1 的访问寄存器的相关定义*/
#define ACCESS(1)       (0x0000 | ((1) << 12))          /*  访问寄存器  */
#define STS(1)          (0x0018 | ((1) << 12))          /*  状态寄存器  */
#define DATA_FIFO(1)    (0x0024 | ((1) << 12))          /*  数据寄存器  */
#define DID_VID(1)      (0x0F00 | ((1) << 12))          /*  设备寄存器  */
/*访问位* /
#define ACCESS_ACTIVE_LOCALITY          0x20    /*读*/
#define ACCESS_RELINQUISH_LOCALITY      0x20    /*写*/
#define ACCESS_REQUEST_USE              0x02    /*写*/
/*状态位* /
#define STS_VALID                       0x80    /*读*/
#define STS_COMMAND_READY               0x40    /*读*/
#define STS_DATA_AVAIL                  0x10    /*读*/
#define STS_DATA_EXPECT                 0x08    /*读*/
#define STS_GO                          0x20    /*写*/
```

初始化函数主要实现如下：

```
int init()
{    for (i = 0; i< 5; i++)
```

```
        /*强制关闭所有的 Locality*/
        write8(ACCESS_RELINQUISH_LOCALITY, ACCESS(i)) ;
    if( request_locality(0) < 0)      return 0;              /*请求使用 Locality 0*/
        vendor = read32(DID_VID(0));                          /**获取厂商 ID/
    if ((vendor & 0xFFFF) == 0xFFFF)     return 0;      /*检查 ID*/
}
```

4.2.3 向 TPM 设备发送命令

函数 Send(unsigned char * buf, int len)实现向 TPM 设备发送命令的功能。在向 TPM 发送命令之前，首先必须确保当前所占用的 Locality 能够访问到 TPM。一旦获得了所请求的 Locality 的控制权之后，就通知 TPM 开始接收命令，然后将命令写入数据缓冲区，写入的数据总量不能超过计数器中规定的数据块大小。每一次命令写入后都需要检查状态寄存器，看 TPM 是否能够继续接收数据。以内存访问方式向 TPM 发送命令的的流程如图 4.6 所示。

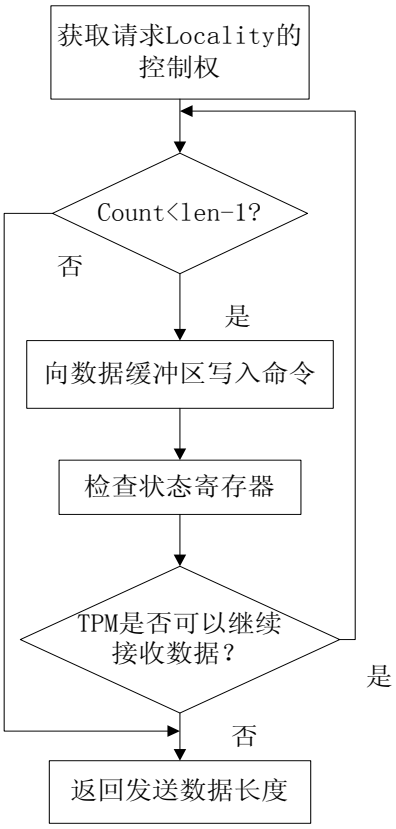

图 4.6 内存访问方式向 TPM 发送命令流程图

4.2.4 TPM 设备接收数据

TPM 接收一个响应包括两个步骤：从 TPM 得到响应数据和对响应数据进行分析。当 TPM 得到响应的数据后，产生中断或者以轮询模式或者状态位有效并且数据可用位为 1 时，调用响应数据分析函数。该函数首先读取响应数据的前 6 个字节，这 6 个字节以网络字节方式存放，对应着 TPM 数据包的首部和整个响应数据包的长度，当把响应数据包的长度值转换为本地字节序方式后，就继续读取剩下的响应数据。一旦读取了全部的响应数据之后，就可以通过设置状态寄存器的 command ready 位让 TPM 准备处理下一条命令。如图 4.7 所示为响应函数的实现流程。

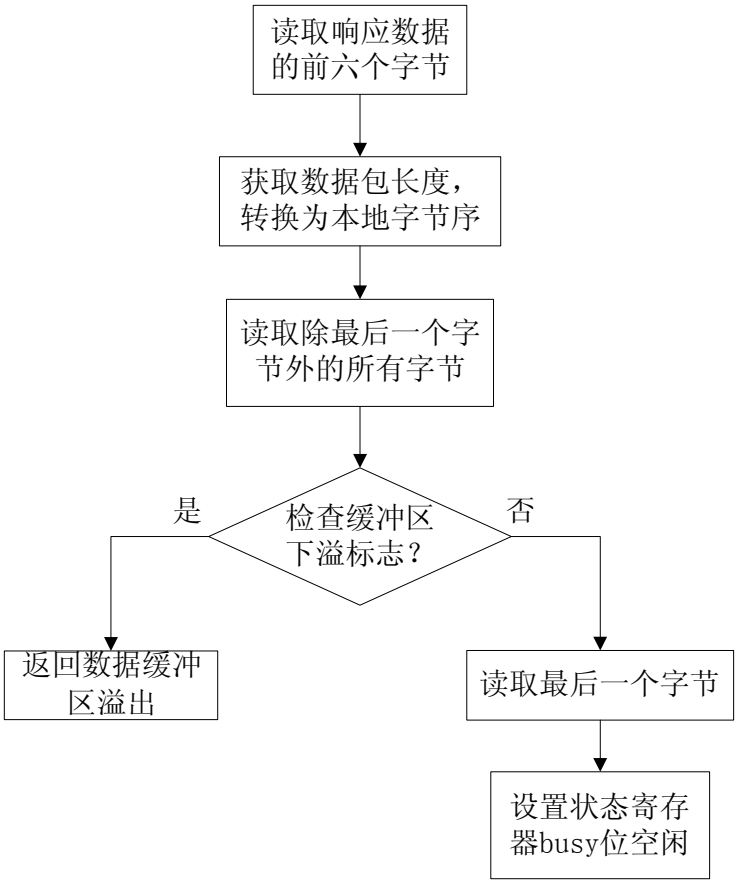

图 4.7　响应数据的实现流程

4.3 底层软件

在设备驱动程序启动之后而完整的 TPM 软件协议栈启动之前，可以通过 BIOS 和 TDDL 库与底层芯片通信。

4.3.1 BIOS 和可信任平台模块之间的通信

当一个操作系统被加载后，设备驱动程序处理与 TPM 的通信。在引导操作系统的过程中，引导加载程序必须在每一步执行之前度量所有的启动模块和操作系统代码。在启动过程中，是没有任何设备驱动程序可用的。为了在启动过程中能更好的与可信平台模块 TPM 进行通信，可信任计算机组 TCG 定义了一个引导加载程序能够调用的最小 BIOS 中断集合，这就提供了一个简单易用的接口。这个接口使用在 Pre-boot 环境下。该接口定义了 INT 1Ah 中断，该中断允许调用者直接访问 TSS 的一个有限集以及访问 TPM。

引导加载程序所必须执行的 TPM 的功能是：在把控制权交给后续的代码和配置数据之前，必须先对它们进行度量。因此，引导加载程序通常将它自身的一部分执行代码进行度量并将度量结果扩展到 PCR4 中，将它自身的配置文件的度量值结果扩展到 PCR5 中，把对操作系统的度量结果扩展到 PCR8 中。

在启动过程中，使用 TPM 的 BIOS 调用的函数有：

（1）TCG_StatusCheck：检查当前 BIOS 是否满足 TCG BIOS 的要求，如果满足则返回当前版本号。

（2）TCG_HashLogExtendEvent：计算输入数据的散列值、创建时间日志并将散列值扩展到某个 PCR 中。

（3）TCG_PassThroughToTPM：给 TPM 发送一个命令字节流。

函数 TCG_StatusCheck 用于验证当前 BIOS 是否支持 TPM 调用，必须在调用 BIOS 的其他 TCG 功能之前先运行这个函数。TCG_HashLogExtendEvent 函数用于先计算数据的散列值，然后将散列值扩展到特定的 PCR 中，最后记录事件日志。对于任何一个 TPM 的请求包都可以通过 TCG_PassThroughToTPM 命令发送给 TPM。然而，这个过程比较复杂，因为引导加载程序需要自己处理所有的请求和响应数据包。TCG_HashLogExtendEvent 函数不同，它会自己处理这些数据包。

4.3.2 TDDL 与 TPM 之间的通信

在设备驱动程序加载后，在所有的 TCG 软件栈服务可用之前，操作系统进行初始化时通常会有一段较短的时间。在这一操作系统初始化过程中，TDDL 接口用来

执行任何必要的 TPM 操作，如装载和解封支持可信启动的密钥。TDDL 库提供在整个 TSS 还未加载完毕时，上述过程中所需的 TPM 的操作。

（1） 启用和清空 TPM

BIOS 是负责启用和清空 TPM 的。当计算机上电之后，TPM 被激活，但是还没有被启动。随后 BIOS 会发出一条 TPM_Startup 命令。这个命令可以完成以下三件事之一：停用 TPM；对 PCR 寄存器进行复位并启动 TPM；从所保存的状态中恢复 PCR 的值并启动 TPM。如果 BIOS 停用 TPM，它将保持不活动状态直到下一个上电周期的到来。由于私密性的原因，没有软件命令能够重新启用 TPM。在计算机启动时清空 TPM 是在启动时完成的，因此，所有的 PCR 的值在启动过程中都要保证正确计算。TPM 的设备驱动程序复杂在挂起时刻发出 TPM_SaveState 请求，以确保有效的 PCR 值在重启时可用。

BIOS 还负责执行 TPM_ForceClear 这一清空命令。这一命令实现对 TPM 的完全重启，卸载所有密钥和句柄，并清除 SRK 和所有者的私密授权信息。

由 BIOS 控制的 TPM 的启用和清除操作是在进入 BIOS 界面设置的。如下图 4.8 所示为 BIOS 中控制 TPM 的界面。

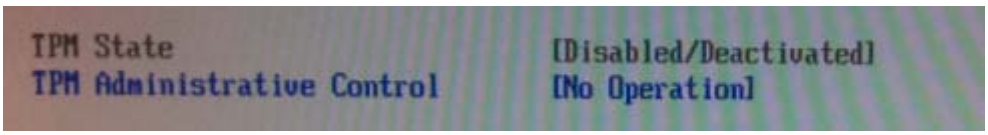

图 4.8　BIOS 界面中对 TPM 的操作

由图 4.8 可以看出，当前 TPM 的状态，并且可以通过“TPM Administrative Control”选项选择启动(Enabled)、禁用(Disabled)、清除(Clear)TPM 的操作。

（2） 获得所有权

想要拥有一个完整性功能的 TPM，还需要获取可信平台模块的所有权。命令 TPM_TakeOwnership 完成这一任务。这个命令有以下四个主要的功能：设置由所有者提供的所有者授权密钥；创建存储根密钥 SRK；设置由所有者提供的 SRK 授权秘密；返回给所有者 SRK 的公用部分。

TPM_CreateWrapKey 命令利用 TPM 硬件模块在芯片内产生一个新的 RSA 密钥。密钥分为签名、加密/解密密钥两种。TPM 不允许用签名密钥进行加密或者用加密密

钥进行签名[40]，因为这样会导致攻击的发生。一个密钥可以可选地设置一个授权，这个授权是将来使用这个密钥所需的。另外，密钥可以与制定的 PCR 值进行绑定。如果是这样做的，那么认证的数据和指定的 PCR 数据必须匹配才能使用该密钥。所有的密钥必须有一个父密钥——这个父密钥可能是 SRK——在密钥的结构返回给用户之前，父密钥用来加密密钥的私有部分。返回的密钥数据会被用户保留用于之后的装载。

TPM_LoadKey 用于将一个密钥加载到 TPM 内的一片易失的密钥存储区，这条命令需要父密钥的认证口令；一旦密钥被加载，TPM 就会用父密钥对载入的密钥的私钥数据进行解密，以便使用。如果该密钥有一个授权，这个授权在该密钥装载时就不需要验证了，但是对于其后试图使用该密钥进行加密或签名的命令来说，则需要验证这个授权。

由于 TPM 中只有有限的密钥存储空间，所以当一个密钥不再被使用时，它必须通过命令 TPM_EvictKey 收回，以便存储空间可以供其他密钥使用。

TPM_Sign 命令用一个载入的密钥来签名当前的数据，通常是签名实际数据的散列值。

TPM_Seal 命令使用 RSA 对数据加密，该命令需要一个装载的加密密钥和该密钥的任何授权密码。TPM_Seal 也可以在封装时指定要使用的 PCR 值。如果将来解封时 PCR 的值不匹配，那么解封就会失败。TPM_Seal 同时将用户提供的授权值应用于封装数据。这样，为了解封数据，用户可能需要封装密钥的口令和封装数据的口令，并且 PCR 的值可能需要匹配。TPM_Unseal 命令用于执行相应的解封操作。

TPM 不提供测试所有者是否存在或者检查正确口令的命令，因此必须运行一些其它的命令，根据返回代码推断 TPM 的状态。

（1）TPM_Reset 命令，即使在 TPM 失效的时候也可以成功的执行，因此可以用该命令检查 TPM 是否存在。

（2）TPM_PcrRead 命令，在 TPM 有效的时候，该命令都可以成功的运行。因此，可以用它来检查 TPM 是否有效

（3）TPM_CreateWrapKey 命令，通过尝试使用该命令和检查错误代码来推断其

所有者及其口令。当命令返回 TPM_NOSRK 时，表示没有所有者；当返回 TPM_AUTHFAIL 时，表示所有者使用的是一个未知的口令。

4.4 本章小结

本章详细介绍了 TSS(TPM Software Stack)体系结构下底层编程接口。首先详细介绍了 TPM 设备驱动程序在 I/O 端口方式通信和内存映射方式通信的流程，并实现了设备驱动程序与 TPM 设备的直接通信。接着介绍了在设备驱动程序启动之后，完整的 TSS 服务启动之前，底层 BIOS 和 TDDL 库分别与 TPM 设备通信的方式。

5 设备驱动程序和 BitLocker 的验证

为了能够满足和 TPM 芯片的直接交互，开发了 TPM 的设备驱动程序。该程序的开发是基于 Atmel 1.1b 芯片，开发环境是在 Linux 下进行的。

BitLocker 驱动器是 Windows Vista 企业级系统和 Windows Server 2008 操作系统中的数据加密功能。它提供增强型的数据保护的功能，防止数据偷窃或在丢失或被盗的计算机上公开，确保在受 BitLocker 保护的计算机解除授权时执行更安全的数据检测。

在配置了 TPM 的计算机上，每次启动该计算机时，每个早期启动组件(如 BIOS、MBR 引导扇区和启动管理器代码)都会检查要运行的代码，计算哈希值，然后将值存储在 TPM 中。一旦存储在 TPM 中，该值就不可替代，除非系统重启。将记录这些值的组合。Windows 下的 BitLocker 借助 TPM 验证早期启动组件的完整性，防止试图将恶意代码插入到这些组件中的恶意攻击。

5.1 设备驱动程序的验证

本文中的设备驱动程序主要实现了 TPM 设备的初始化以及 TPM 收发命令的功能。基于以上功能对驱动程序进行功能测试。

5.1.1 测试目的

对该设备驱动程序测试和改进，以达到和底层 TPM 芯片正常通信的目的。该驱动程序性能的优劣将直接影响到数据收发速度。

5.1.2 测试环境

表 5.1 测试硬件软件环境

项目	具体环境及版本
硬件环境	Intel（R）Core(TM)2 Duo CPU T5450 1.66Ghz 2GB 内存，Atmel TPM 芯片
操作系统	Red Hat 5
固件要求	TCG UEFI BIOS

5.1.3 测试步骤

在 Linux 命令行窗口，编译 tpm.c 文件，能看到窗口打印出的信息提示 Atmel 的 TPM 芯片被检测到，向 TPM 芯片发送命令和接收命令的函数正常运行。如图 5.1 所示。

```
[root@idcserver tpm]# g++ tpm.c -o tpm
[root@idcserver tpm]# ./tpm
Atmel TPM device detected.
Send function run normally.
Receive function run normally
[root@idcserver tpm]#
```

图 5.1 设备驱动程序执行结果

5.1.4 测试结果分析

该驱动程序的初始化工作主要实现对 TPM 芯片设备厂商的检测，由图 5.1 可以看出，最终设备的厂商 ID 被检测到，则基本功能实现。核心函数的收发功能也正常运行。

5.2 BitLocker 驱动器功能验证

BitLocker 驱动器是 Windows server 2008 提供的数据保护功能。但是它必须依赖于主板上 TPM 芯片的集成和 EFI BIOS 对 TCG 的支持。对于它的验证集中于功能测试。

5.2.1 测试目的

主要为了测试 BitLocker 驱动器在装配了 TPM 的主板上功能的完整性，以及流程是否正确可靠。因此系统采用黑盒测试，主要集中于功能测试。

5.2.2 测试环境

（1）硬件要求

若要使 BitLocker 利用 TPM 提供的系统完整性检查，硬件平台必须配置 1.2 版的芯片。

（2）固件要求

具有 TPM 的计算机还必须具有符合受信任计算组(TCG)的 BIOS。BIOS 为预操作系统启动建立了一个信任链，并且必须包括对指定的 TCG 信任度量静态根的支持。

（3）软件要求

使用具有 BitLocker 驱动加密功能的 Vista 系统或者是 Windows Server 2008 系统。并且至少将硬盘分区为两个卷：操作系统卷(或启动卷)包含 Windows Vista 操作系统及其支持文件；必须使用 NTFS 文件系统对其进行格式化。在此卷上启用 BitLocker；

系统卷包含 BIOS 已启动平台之后加载 Windows 所需的文件。在此卷上不启用 BitLocker。若要使 BitLocker 起作用，请不要对系统卷进行加密，必须区别于操作系统卷，并且必须使用 NTFS 文件系统对其进行格式化。系统卷应该至少为 1.5 千兆字节(GB)。

5.2.3 测试步骤

（1）UEFI BIOS 对 TPM 的支持

计算机上电后，按“F2”进入 BIOS 界面。在 Security 子菜单下，可以看到 TPM 的状态项和配置项，参照图 5.2 所示。在 Security 子菜单下，可以看到当前 TPM 的状态为“Disabled/Deactivated”，如果要启用 BitLocker 的功能，必须要设置 BIOS 下的管理员账号。只有管理员身份才能实现对 TPM 的启用和禁用。当管理员密码设置后，可以在 BIOS 界面下设置选项“TPM Administrative Control”为“Enabled”。

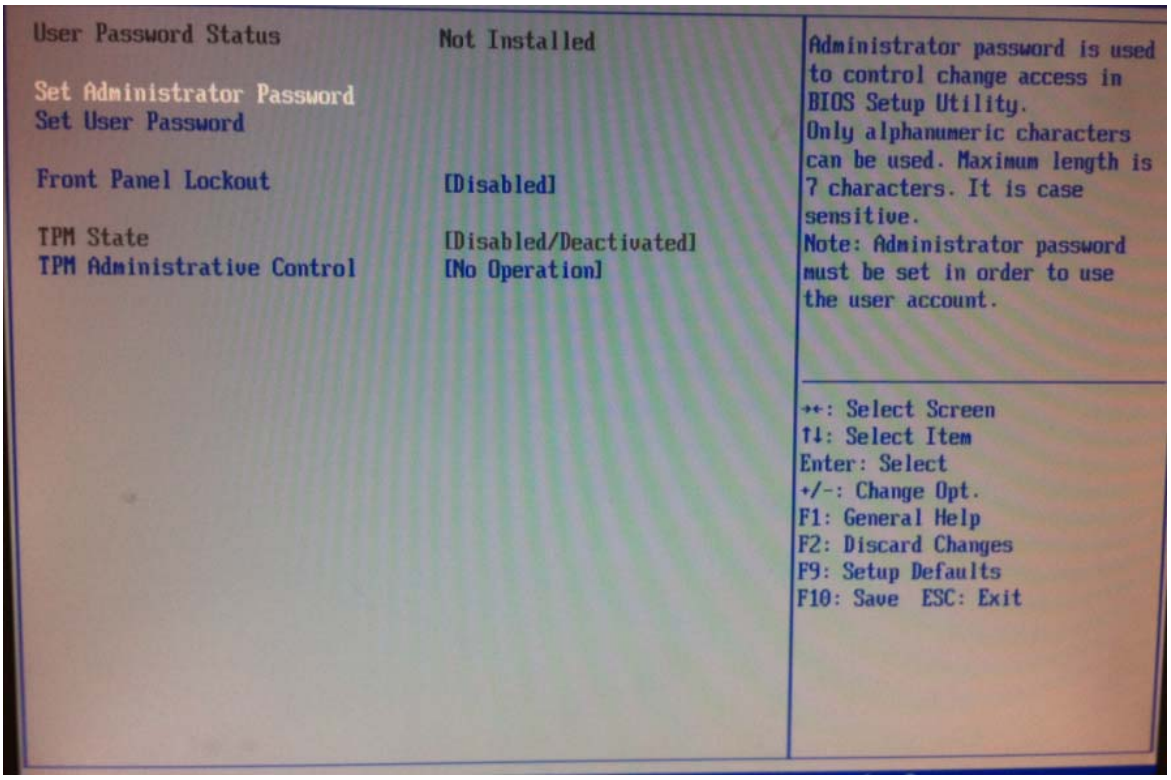

图 5.2 TPM 在 BIOS 下的设置

（2）BitLocker 驱动加密器安装

在 Windows Server 2008 中可以对可选功能 BitLocker 驱动器进行安装以完成对整个硬盘的加密过程。可以选择以管理员身份进入命令提示符窗口，运行 manage-bde.exe –install 命令来安装 BitLocker 驱动器。当安装完成后，会提示重启计算机。

（3）硬盘加密

当 BitLocker 驱动加密器安装成功之后，需要以管理员身份进入 BIOS 设置界面启动 TPM，并且获得使用者授权。请参照上一章中图 4.8。当 TPM 的状态为启动(Enabled)、激活(Activated)、授权(Ownership has been taken)时，可以在操作系统的控制面板下面打开 BitLocker 驱动加密器，发现当前硬盘的状态均是未加密状态(Off)。此时可以选择需要加密(Turn on BitLocker)的硬盘。当你选择 Turn On BitLocker 时，系统会提示用户保存系统恢复密码，如下图 5.3 所示。用户可以选择将恢复密码保存在 USB 设备中；保存在文件中；或者打印出恢复密码。

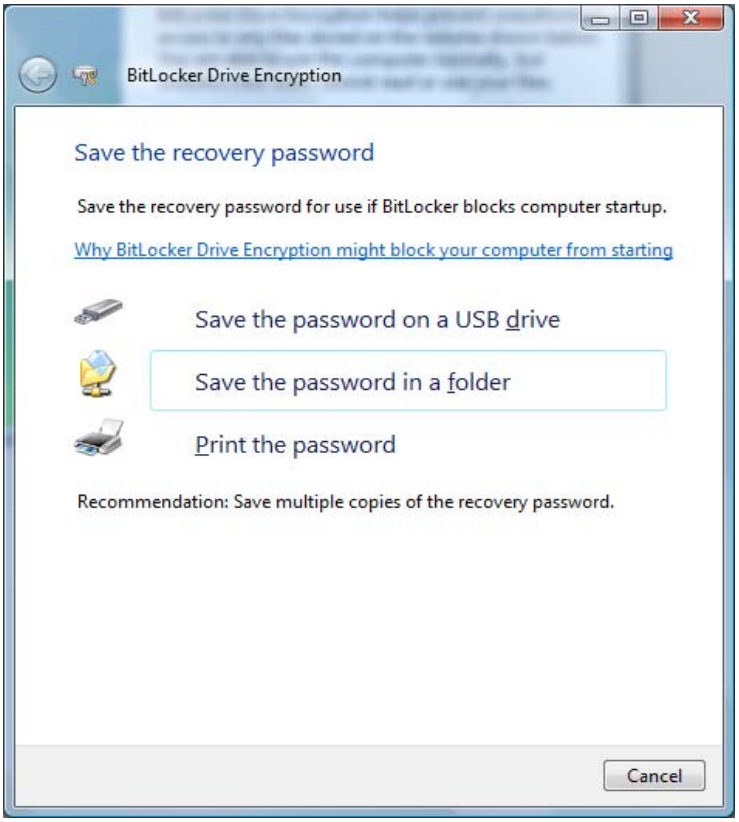

图 5.3　保存恢复密码

如果将恢复密码保存在 USB 设备中，BitLocker 驱动加密器开始对所选硬盘进行全盘加密。完成加密需要较长的一段时间，当加密完成后，可以看到硬盘的状态为 On，此时用户的可选操作就是解密硬盘(Turn Off BitLocker)。

（4）硬盘访问保护

当下一次用户想要进入加密硬盘所在的操作系统时，如果存放恢复密码的 USB 设备没有被系统检测到，系统将会提示插入存储介质或输入恢复密码。如图 5.4 所示。

图 5.4　提示输入恢复密码

当重新插入存放恢复密码的设备于系统上时，重启计算机后，系统将会提示恢复密码被检测到，系统安全启动到操作系统下。或者可以选择不重启计算机，手动输入恢复密码，启动系统。

5.2.4　测试结果分析

通过测试，发现可信 UEFI BIOS 对 TPM 支持良好，Windows Server 2008 下的 BitLocker 驱动器的数据保护功能良好。

5.3 本章小结

本章主要介绍了 TPM 设备驱动程序的执行结果和 Windows Server 2008 操作系统下的 BitLocker 驱动加密器的验证测试，通过在操作系统下的 BitLocker 驱动器结合 TPM 芯片和 TCG BIOS 共同验证了这一安全方案在 Windows Server 2008 下的实现。

6 总结与展望

6.1 全文总结

计算机安全是计算机领域的重要课题之一，基于 UEFI 的信任链设计是从底层固件层面引入可信平台模块 TPM 对系统上电之后的启动过程建立信任链，通过对每一个执行过程的完整性度量来保护计算机系统的可信，有效的防止了软件的漏洞给计算机带来的严重影响。对于信任链底层与 TPM 芯片的通信，本文实现了基于 TPM1.0b 和 TPM1.2 标准的设备驱动程序。并对驱动程序进行了验证。对于 Windows 2008 提供的数据保护功能 BitLocker 驱动器在可信计算机系统中的应用也得到了验证。

在对基于 UEFI 的信任链设计及 TPM 驱动程序实现的研究中，本文主要做了如下几个工作：

（1） 分析相关技术

分析了 UEFI BIOS 的平台初始化框架，通过与传统的 Legacy BIOS 的相比，新一代 UEFI BIOS 在可扩展性和兼容性方面更具优势，安全性得到了更好的保障。为数据增强功能的实现提供了固件层的保证。根据可信计算机系统的组成，掌握基本的信任链的建立过程以及可信计算平台的核心部件功能。

（2） 信任链设计

通过分析可信计算平台的硬件组成和 UEFI 运行阶段的特点，确立了由 TPM 芯片和 BIOS 协同组成的可信度量根。将可信计算的基本思想引入到系统上电之后的启动过程，完成对每个执行阶段的完整性度量，以决定是否将系统的控制权交由下一个执行过程。通过这种可信关系的传递，建立基于 UEFI 下的信任链和平台完整性度量机制。

（3） 底层设备编程接口

从整个软件架构的最底层开始，首先具体介绍了 TPM 设备驱动程序的在 I/O 端口通信和内存访问方式通信的实现方法。接着介绍了 BIOS 和 TPM 之间是如何进行通信的以及 TPM 设备驱动库和 TPM 之间的通信方式。

（4） TPM 设备驱动程序的验证和 BitLocker 的应用

介绍了 TPM 设备驱动程序的测试目的、测试环境、测试步骤和测试结果分析。通过对 TPM 1.0b 的通信证明驱动程序所实现的功能良好。对 BitLocker 的应用，通过 TPM 芯片，可信 UEFI BIOS 固件，Windows 2008 中的 BitLocker 驱动加密器的协作构建操作系统下的可信关系，实现对硬盘数据的加密和系统恢复的功能。

6.2 展望

本文在计算机系统安全增强方案的研究中应用了一些比较普及和成熟的技术进行实现。但是，由于时间、经费一级各方面条件的限制，本方案还是有一些可以改进的地方的。在今后的学习或工作中，还是值得我去做进一步的研究。

（1）由于 TPM 芯片在个人电脑上的普及度并不高，有待于研究类似于 U 盾之类的固件层数据保护方案，通过这样的模式来普及该安全增强型技术，使得广大计算机使用者都能受惠于该技术。

（2）目前该功能仅仅在 Windows Vista 和 Windows 2008 以上系统下得到了有效的应用，但是大多数的服务器都都是运行于 Linux 操作系统下，有待于将该技术实现于 Linux 操作系统下。

致 谢

时光如梭，短暂而充实的两年半的研究生生涯即将结束。在此，我由衷的像给予我极大帮助的苏曙光导师和学院其他老师，同学，同事，父母表示我衷心的感谢。

首先，我要感谢我的导师苏曙光老师，从研一的项目指导，研二的实习推荐，到后来论文的选题，修改，成型，苏老师都给了我很多的意见，这对我一步步的成长起到了渐进式的推动作用。相信没有他的谆谆教导，我是不可能成长的这么快的。他在科研上的学术精神和钻研精神将对我今后的工作和生活终身受用。在此，我衷心的送上我的敬意。

其次，我要感谢我们软件学院所有的领导和老师。因为他们的辛勤耕耘为我们提供了优质的学习环境和氛围。没有他们的辛勤培育，我们是不可能从一名稚嫩的本科生蜕变成一名成熟的研究生。在这里，我完成了我人生中重要的质变。

再则，我还要感谢我的同学、师兄、师姐、同事。因为在和他们的相处中，我从大家的身上学到了很多自己并不具备的品质，他们也为我的项目与工作给予了很多建设性意见。

最后，我尤其要感谢我的父母和家人的鼓励和支持。在我学习和工作中，总会遇到一个又一个的难题，是他们无私的关怀和呵护使我闯过了一个又一个的难关。

参考文献

[1] 徐拾义. 可信计算系统设计和分析. 北京: 清华大学出版社, 2006: 25-26

[2] 陈文钦. BIOS 研发技术剖析. 北京: 清华大学出版社, 2001: 57-58

[3] Xi Zhezhang, Yong Xie, Yao Hong etc. A Multi-core Security Architecture Based on EFI. Beijing: Publishing House of Electronic Industry, 2008: 35-40

[4] 章睿, 刘吉强, 彭双和. 基于 EFI 信任链传递研究与实现. 计算机应用, 2007, 12(9): 24-25

[5] 黄楠, 何杰, 刘峰等. 基于 TCG 构建的可信终端. 计算机与数字工程, 2006, 34(4): 74-76

[6] 蒋逸明, 孙元浩, 张申生. 基于 EFI 的图行界面的设计与研究. 计算机工程, 2007, 9(14): 70-72

[7] 王海波, 张申生. EFI/TIANO 下的图形界面新方案 NUWA. 微型电脑应用, 2007, 4(10): 97-99

[8] 石俊菁. EFI 接口 BIOS 驱动体系的设计实现与应用: [硕士学位论文]. 南京: 南京航空航天大学图书馆, 2006

[9] 熊毅. 基于可扩展固件接口(EFI)的高安全 BIOS 的研究与实现: [硕士学位论文]. 北京: 中国科学院研究生院, 2007: 25-27

[10] 潘登, 刘光明. EFI 结构分析及 Driver 开发. 计算机工程与科学, 2006, 28(2): 20-22

[11] 任超, 黄林鹏. 基于 EFI 上的 USIM 设备驱动程序开发. 计算机工程, 2004, 9(30): 27-28

[12] 倪志欣. 如何有力支持 EFI 平台. 中国计算机报, 2006, 3(15): 15-17

[13] 周伟东, 许榕生. 基于 EFI BIOS 的安全 Agent 设计. 北京电子科技学院学报, 2007, 5(4): 22-25

[14] 谢勇, 来学嘉, 邓子健. EFI 及其安全性研究. 信息安全与通信保密, 2007, 7(8): 226-279

[15] 谢勇. 基于 EFI 双核的安全系统框架设计与研究: [硕士学位论文]. 上海: 上海交通大学图书馆, 2008

[16] 徐娜. 基于可信赖计算平台的可信执行环境研究与实现: [硕士学位论文]. 北京: 中国科学院研究生院, 2006: 19-21

[17] 张焕国. 可信计算研究进展. 武汉大学学报, 2006, 52(5): 513-518

[18] Trusted Computing Group. TCG Software Stack Specification, Version 1. 20. 2006: 15-18

[19] 赵宇. 基于 TPM 规范的 HMAC/SHA-1 IP: [硕士学位论文]. 上海: 上海交通大学图书馆, 2007

[20] Trusted Computing Group. Trusted Platform Module Main Specification, Version 1. 2, 2006: 147-151

[21] 泰戈, 韩文报. 关于可信计算平台模块的研究. 信息工程大学学报, 2006, 7(4): 197-198

[22] 沈昌祥. 可信计算平台和安全操作系统. 网络安全技术与应用, 2005, 5(4): 8-9

[23] David Challener, Kent Youder, Ryan Chatherman. A Practical Guide to Trusted Computing. China Machine Press, 2009: 59-60

[24] Sean W. Smith. Trusted Computing Platforms: Design and Application. Qinghua University Press, 2006: 45-60

[25] Trusted Computing Group. Trusted Platform Module Protection Profile, 2004: 25-30

[26] 孔维飞. 可信计算平台的工作原理与应用研究. 武汉科技学院学报, 2003, 8(12): 81-84

[27] Michael Kinney. Solving BIOS Boot Issues with EFI. Intel Developer UPDATE Magazine, 2000: 25-32

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

[29] Najwa Aaraj, Anand Raghunathan, Srivaths Ravi etc. Energy and Execution Time Analysis of a Software-based Trusted Platform Module. Proceedings of the conference on Design, Automation and Test in Europe, 2007(4): 1128-1133

[30] Aarthi Nagarajan, Vijay Varadharajan, Michael Hitchens. Trust Management and Negotiation Conference on Parallel and Distributed Computing, Applications and Technologies, 2007: 452-460

[31] R. Sailer, X. Zhang, T. Jaeger etc. Design and Implementation of a TCG-based Integrity Measurement Architecture. California: 13th Usenix Security Symposium, 2004: 255-261

[32] Boris Balacheff, Liqun Chen, David Plaquin etc. Trusted Computing Platforms: TCPA Technology in Contex. Prentice Hall PTR, 2002: 168-175

[33] Xinwen Zhang, Onur Aciicmez, Pjean-pierre Seifert. A trusted mobile phone reference architecture secure kernel. Proceedings of the 2007 ACM workshop on Scalable trusted computing, 2007(11): 7-14

[34] Jason Reid, Juan M. Gonzlez Nieto, Ed Dawson etc. Privacy and Trusted Computing. 14th International Workshop on Database and Expert Systems Applications(DEXA'03), 2003: 81-85

Saltier J H, Schroeder M D. The Protection of Information in Computer Systems. Proceedings of the IEEE, 1995, 63(9): 1278-1282

[35] Rui Zhang, Jiqiang, Shuanghe Peng. A Trusted Bootstrap Scheme on EFI. International Conference on Multimedia Information Networking and Security, 2009: 200-204

[36] 王新成. 可信计算与系统安全芯片. 计算机安全, 2005, 5(10): 355-360

[37] 邢启江, 肖政, 侯紫峰等. 一种基于 TPM 芯片的计算机安全体系结构. 计算机工程, 2007, 33(15): 152-155

[38] 谭良, 周明天. 可信操作系统研究. 计算机应用研究, 2007, 24(12): 10-114

[39] 谭兴烈. 可信计算平台中的关键部件 TPM. 信息安全与通信保密, 2005, 12(3): 511-514

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

[41] Menasce. Security Performance. IEEE Trans. On computer safety, 2003, 7(3): 84–87