



学 号： 22620121018

论 文 密 级： 公开

中图分类号： TP309

学科分类号： 520

学 校 代 码： 90005

信息工程大学

硕士学位论文

基于 USB Key 的 系统可信启动技术研究

论 文 作 者： 贾天江

指 导 教 师： 曾光裕

申 请 学 位： 工学硕士

学 科 名 称： 计算机科学与技术

研 究 方 向： 信息安全

论文提交日期： 2015 年 4 月 20 日

论文答辩日期： 2015 年 6 月 17 日

信息工程大学网络空间安全学院

二〇一五年四月

**A Dissertation Submitted to
PLA Information Engineering University
for the Degree of Master of Engineering**

Research on Trusted Boot Technology Based on USB Key

Candidate: Jia Tianjiang

Supervisor: Zeng Guangyu

Apr. 2015

原创性声明

本人声明所提交的学位论文是本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，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得信息工程大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并标示谢意。

学位论文题目： 基于 USB Key 的系统可信启动技术研究

学位论文作者签名： _____ 日期： _____ 年 _____ 月 _____ 日

作者指导教师签名： _____ 日期： _____ 年 _____ 月 _____ 日

学位论文版权使用授权书

本人完全了解信息工程大学有关保留、使用学位论文的规定。本人授权信息工程大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将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、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目： 基于 USB Key 的系统可信启动技术研究

学位论文作者签名： _____ 日期： _____ 年 _____ 月 _____ 日

作者指导教师签名： _____ 日期： _____ 年 _____ 月 _____ 日

保密级别： _____ 保密年限： _____ 年 _____ 月至 _____ 年 _____ 月

（保密委员会公章）

摘 要

计算机的启动过程是终端系统运行过程中的关键环节之一，如果该过程受到攻击，那么在内核层和虚拟层中的各种安全机制也将失效。因此，开展系统启动过程的安全防护研究具有十分重要的现实意义。

本文在分析研究可信计算理论、UEFI BIOS 架构和运行机制、终端启动过程及其脆弱性的基础上，设计实现了基于 USB Key 的可信启动系统，并对该系统的功能和性能进行了测试。

论文的主要工作和贡献包括：

1. 设计了基于 USB Key 的可信启动系统框架，并提出了一种高效的混合信任链构建过程。在深入研究 UEFI BIOS 架构和运行机制及其脆弱性的基础上，依据可信计算理论提出了基于 USB Key 可信启动的系统框架。然后，根据启动过程中各个阶段的运行特点，提出了一种混合信任链传递过程：在 UEFI BIOS 运行阶段采用链式信任链构建方式，以实现细粒度安全度量，保证密切联系的 UEFI BIOS 各运行阶段的完整性；在操作系统引导阶段则采用星形结构的信任链构建方式，以降低信任损耗，提高系统可信性。

2. 设计实现了基于 UEFI 驱动模式的 USB Key 驱动。通过对 UEFI BIOS 驱动模式的研究以及对 USB 驱动栈的分析，分别实现了 EFI_DRIVER_BINDING_PROTOCOL 协议中的 Supported()、Start()和 Stop()三个函数以及其他相关协议，实现了 UEFI BIOS 环境下的 USB Key 驱动，并利用中断传输和控制传输，实现了 UEFI 环境下 USB Key 设备的 I/O 抽象。

3. 设计实现了基于 USB Key 的完整性度量和认证功能。通过对 USB Key 命令系统的研究，尤其是命令应答机制和命令处理流程，实现了基于 USB Key 的完整性度量模块和双因子身份认证模块，分别用以完成系统的完整性度量、用户的合法性验证以及 USB Key 的唯一性验证功能。

4. 提出了一种基于白名单的硬件检测机制。白名单中存储了已知的硬件的设备名，和其相应硬件接口的 hash 标准值。在启动过程中，当系统检测到有硬件设备连接在系统上时，首先检测该硬件设备是否在白名单中；然后利用 UEFI 中的硬件驱动收集连接在主机上的硬件设备接口信息，并对该信息进行 hash 运算；最后，通过将这些 hash 值与白名单中的标准值比较，检测该硬件是否为可信硬件。硬件白名单存储于基准库中，并通过 USB Key 设备确保基准库的安全性。

5. 构建了基于 USB Key 的可信启动系统。首先，为了确保系统的完整性，建立了基于 UEFI Capsule 机制的恢复更新模块，用于在系统检测出现问题时对系统进行恢复。然后，联合驱动模块、度量模块以及认证模块构建了基于 USB Key 的可信启动系统。测试结果表

明，本系统能有效阻止对启动过程的篡改和非法用户的入侵，并具有较小的时间开销。

关键词：UEFI BIOS，USB Key，完整性度量，身份认证，可信启动，可信计算

Abstract

The start-up process of computer is a key link of the terminal system operation process, if the procedure was tempered with, then various security mechanisms in kernel or virtual level will become ineffective. Therefore, the research on the security of start-up process is of great practical significance.

According to the analysis of trusted computing theory, the architecture and operating mechanism of UEFI BIOS, terminal startup process and its vulnerability, a trusted boot system based on USB Key is designed and implemented, and its function and performance are tested.

Major contributions and innovations in this thesis are as follows:

1. A trusted boot system framework based on USB Key is designed and an efficient mixing process chain of trust is proposed. On the basis of in-depth research and analysis of UEFI BIOS architecture, operational mechanism and its vulnerabilities, a trusted boot system framework based on USB Key is designed according to trusted computing theory. Then, according to the operating characteristics of the various stages of the boot process, a mixing trust chain transfer process is presented: In UEFI BIOS running stage, chain type of trust chain is adopted to achieve finer-grained security metrics; in the operating system boot stage, the star-shaped structure of trust chain is adopted. In this way, the system can avoid loss of trust and enhance the credibility.

2. Design and implement a USB Key driver based on UEFI driver model. Based on analysis of UEFI BIOS driver model and the USB driver stack, the Supported (), Start () and Stop () of the EFI_DRIVER_BINDING_PROTOCOL and other related protocols are realized, and then the USB Key driver in UEFI BIOS environment is realized.

3. Realize the integrity measurement and attestation function based on USB Key. Through the study on USB Key command system, especially command response mechanisms and command processing procedure, and taken advantage of security protocols in UEFI, we realize the integrity measurement module and two-factor authentication module based on USB Key, in order to complete to verify the integrity measurement of the system, the legitimacy of users and the uniqueness of USB Key respectively.

4. This paper proposes a hardware detection mechanism based on white list. What stored in the white list are known hardware devices' names, and hash values of their corresponding hardware interfaces. In the process of start-up, firstly, the system verifies whether the hardware in the white list or not when it detects a hardware device, secondly the system collects the

interface information of the hardware linked on the terminal host by taking advantage of hardware drivers in UEFI, then calculates the hash value of the information. Finally, the system test whether the hardware is reliable or not by comparing hash value with the standard values of white list. White list stored in the baseline library which is secured by the USB Key equipment.

5. A trusted system based on USB Key is constructed. First of all, in order to ensure the integrity of the system, the restore update module based on UEFI Capsule mechanism is realized, which can use to recover the system when some problems arises during verifying process. Integrated driver module, measurement and authentication module, the trust boot system based on USB Key is built. Actual test verifies that the system can effectively prevent tampering with the start process and invasion of illegal users, also it has a small time overhead.

Key words: UEFI BIOS, USB Key, Integrity Measurement, Identity Authentication, Trusted Boot, Trusted Computing

目 录

摘 要	I
Abstract	III
图 录	IX
表 录	XI
第一章 绪 论	1
1.1 课题背景	1
1.2 课题研究的意义	2
1.3 国内外研究现状	4
1.3.1 可信计算研究现状	4
1.3.2 安全启动研究现状	5
1.4 论文研究内容	7
1.5 论文结构安排	8
第二章 论文相关理论基础	11
2.1 可信计算概述	11
2.1.1 可信计算简介	11
2.1.2 可信平台模块分析	11
2.2 统一可扩展接口	13
2.2.1 UEFI 简介	13
2.2.1 UEFI BIOS 框架	14
2.3 UEFI 关键组件	15
2.3.1 启动管理器	15
2.3.2 UEFI 核心固件	16
2.3.3 UEFI 映像	19
2.4 UEFI 驱动模式	20
2.5 UEFI 运行机制	21
2.6 本章小结	23
第三章 基于 USB Key 的可信启动架构设计	25
3.1 安全需求分析	25
3.2 设计原则与设计思路	26
3.2.1 设计原则	26
3.2.2 设计目标	26

3.2.3	设计思路	27
3.3	基于 USB Key 的可信启动系统框架	30
3.3.1	基于 USB Key 的可信启动架构	30
3.3.2	基于 USB Key 的可信启动过程	31
3.3.3	系统安全性与可靠性分析	33
3.4	关键问题的提出	34
3.4.1	UEFI 下 USB Key 的驱动	34
3.4.2	基于 USB Key 的完整性度量与认证	34
3.4.3	标准值的存储与封装	35
3.4.4	度量点的选取	35
3.5	本章小结	36
第四章	基于 UEFI 驱动模式的 USB Key 驱动程序	37
4.1	USB 系统结构	37
4.2	UEFI USB 驱动程序分析	38
4.3	基于 UEFI 驱动模式的 USB Key 驱动程序的设计与实现	39
4.3.1	驱动程序的关键函数	40
4.3.2	EFI_DRIVER_BINDING_PROTOCOL 的实现	40
4.3.1	其它相关协议的实现	45
4.4	本章小结	47
第五章	基于 USB Key 的完整性度量与认证	49
5.1	USB Key 命令系统	49
5.2	UEFI 下 USB Key 应用程序的设计与实现	51
5.2.1	完整性度量相关程序的设计与实现	51
5.2.2	身份认证程序的设计与实现	54
5.3	本章小结	56
第六章	基于 USB Key 可信启动系统的实现与测试	59
6.1	基于 USB key 可信启动系统的实现	59
6.1.1	基于白名单机制的基准库	59
6.1.2	基于 Capsule 更新机制的恢复模块	61
6.1.3	系统实现	62
6.2	系统测试	64
6.2.1	功能测试	64
6.2.2	性能分析	68
6.2.3	对比分析	69

6.3 本章小结	69
结束语	71
致 谢	73
参考文献	75
作者简介	79

图 录

图 1	信任链流程.....	4
图 2	星形信任结构.....	5
图 3	TPM 内部结构示意图.....	12
图 4	UEFI 逻辑结构示意图.....	14
图 5	UEFI BIOS 系统框架.....	15
图 6	UEFI 协议.....	17
图 7	句柄数据库示意图.....	18
图 8	句柄类型及其间的逻辑关系.....	19
图 9	驱动程序的初始化过程.....	21
图 10	UEFI 运行过程.....	22
图 11	USB Key 硬件结构示意图.....	28
图 12	基于 USB Key 的可信启动架构.....	31
图 13	基于 USB Key 可信启动过程.....	31
图 14	UEFI 阶段链式信任链流程图.....	32
图 15	操作系统阶段星形信任链流程图.....	33
图 16	USB 系统框图.....	37
图 17	UEFI USB 驱动栈.....	39
图 18	USBKeyDriverBindingSupported 定义.....	41
图 19	USBKeyDriverBindingStart 定义.....	42
图 20	USBKeyDriverBindingStop 定义.....	44
图 21	EFI_COMPONENT_NAME_PROTOCOL 定义.....	45
图 22	USBKeyGetDriverName 定义.....	46
图 23	静态表.....	46
图 24	UsbKeyGetControllerName 定义.....	47
图 25	命令处理流程.....	49
图 26	命令应答机制.....	50
图 27	命令处理过程.....	50
图 28	hash 计算应用程序入口函数.....	51
图 29	函数处理流程.....	52
图 30	完整性度量过程.....	53
图 31	用户 USB Key 级别鉴定过程.....	55

图 32 身份认证过程 56

图 33 硬件白名单验证流程 60

图 34 Capsule 结构 61

图 35 Capsule 头结构 62

图 36 完整性度量和验证过程 63

图 37 USB Key 驱动程序加载测试 65

图 38 “drivers”指令测试 66

图 39 完整性测试 66

图 40 未插入 USB Key 68

图 41 输入错误 PIN 码 68

表 录

表 1 UEFI 映像类别 19

表 2 USB Key 驱动程序中需要实现协议与函数 40

表 3 完整性度量需要实现的应用程序 53

表 4 standard 基准表 60

表 5 UEFI_SEC_PROTOCOL 62

表 6 测试环境 64

表 7 测试结果 67

表 8 系统对比 69

第一章 绪论

1.1 课题背景

随着计算机技术的快速发展,计算机已经广泛应用于各个领域,在给各行各业带来便捷的同时计算机终端的信息安全也在遭受着越来越严峻的挑战。目前,恶意行为发展十分迅猛,并且它已经可以通过加密、伪装等各种自保护方法来隐蔽自己,在用户毫无察觉的情况下实现对系统信息的篡改、窃取和破坏,给终端系统的安全带来了巨大的威胁,也给各级用户造成了巨大的损失。因此,保护终端系统的安全、确保终端计算环境的可信性和可靠性对于当今社会具有十分重要的意义。

2014 年 4 月,国家计算机网络应急技术处理协调中心(简称国家互联网应急中心,CNCERT 或 CNCERT/CC)发布了《2013 年中国互联网网络安全态势综述》^[1]。报告中指出 2013 年期间我国被攻击、被挟持的主机数量依旧十分庞大,并且具有国家背景的有组织的攻击频发,最具代表性的是美国的“棱镜计划”。斯诺登曝光了多项美国国家安全局的网络监控项目,披露了美国情报机构对多个国家和民众长期实施监听和网络渗透攻击,表明各个国家的信息安全和互联网用户隐私已经面临严重威胁,而我国是监听和攻击的重点。综述中还提到,我国的政府机构、基础电信企业、科研院所以及大型商业机构的网络信息系统频繁遭受攻击和渗透入侵,这严重威胁了我国的关键基础设施和重要信息系统安全。

传统的防护方法,如防火墙、入侵检测等越来越难以适应恶意行为的飞速发展,其原因主要体现在以下几个方面:第一,随着操作系统不断发展,操作系统的代码量越来越大,例如微软已经停止维护 Windows XP 系统,其源代码就已经达到了 4000 多万行^{[2][4]},如此巨大篇幅的代码不可避免地就会存在一些漏洞^[5];第二,恶意软件的数量越来越庞大,且增速越来越快。2014 年 12 月迈克菲发布《迈克菲实验室威胁报告(2014 年 11 月刊)》^[6],该报告包含了对 2014 年第三季度威胁的分析以及对 2015 年威胁态势的预测。报告中指出在第三季度,迈克菲实验室每分钟检测到的新威胁数量就超过了 307 个,也就是说,每秒钟就会出现超过 5 个新的恶意行为。并且这些恶意软件攻击手段呈现出多样化,例如 BIOS 陷门攻击、具有特殊权限的内核级 Rootkit、ROP 攻击等;第三,相对于恶意攻击技术,防护工具的更新具有一定的滞后性,安全工具往往只有在发现系统遭受恶意攻击之后才会发布针对性的更新,然而,此时系统已然遭到了攻击,攻击者已经获得了用户的部分数据信息;第四,当前恶意攻击的目标逐渐向底层靠近,基于操作系统层面的安全防护机制并不能满足不同机构或阶层对终端的安全需求,如 2010 年出现了基于 UEFI(Unified Extensible Firmware Interface,统一的可扩展固件接口)的 Bootkit,该类型 Bootkit 能够感染 bootmgr.efi 文件,绕过 Windows 操作系统的 PatchGuard 保护机制,并在系统启动过程中实现内核劫

持，给计算机安全带来巨大的安全隐患。

由此，为了更好地保护计算机终端安全，解决传统安全防护方法与手段的不足，可以采用可信计算组织^[20](Trusted Computing Group, TCG)提出的增强型安全终端体系结构来保证整个系统的安全。其主要设计思想是在终端设备的硬件平台上引入具有安全存储和加密功能的可信平台模块(Trusted Platform Module, TPM)^[8]，通过 TPM 来确保终端系统的安全性。利用 TPM，可以实现对系统的启动过程、运行阶段进行安全防护，及时发现恶意行为，保障终端平台各组件的完整性和可信性，以确保计算机终端的正常运行，保证整个终端信息系统运行的完备性和可靠性。

但是，在实际运用过程中发现，TPM 是嵌入在系统主板上的，其易用性和适用性具有很大的限制，并且 TPM 在密码配置和密钥管理方面也存在较多的不足^[25]，如密钥种类繁多、授权协议复杂等。而 USB Key 具有与 TPM 类似的硬件构成，并且，USB Key 应用范围更广，更加容易获得，自身的安全性也可以得到保证。因此，本课题采用 USB Key 代替 TPM 实现系统的可信过程。

1.2 课题研究的意义

传统的安全技术大多建立在操作系统应用层或者内核层，这就首先需要信任操作系统自身的行为，同时也要考虑系统内部其它程序的相互作用，这给计算机终端的安全防护带来了一定的局限性。可信计算与传统的安全防护思想相比，更具有主动性，首先确保信任根的安全性，然后通过当前可信的安全组件来组建更大的安全系统。本课题研究利用可信计算的思想，通过在不可信终端上添加可信硬件 USB Key，由可信硬件 USB Key 对终端上的 UEFI BIOS 及操作系统启动过程的各个组件进行完整性度量、验证，从而保证系统启动过程的完整性，防止其被非法篡改和破坏，最终保证它的启动过程可信，从而建立最初的可信赖的计算环境。

课题的研究意义主要体现以下几个方面：

(1) 构建更加灵活、应用更加广泛的可信根

可信计算组织通过利用 TPM 实现其可信启动过程。但在实际运用中发现 TPM 具有一些缺陷，如用户需要在主板上添加 TPM 芯片，并且用户对 TPM 的更换或者扩展不可控等。因此利用 USBKey 作为可信硬件实现 TPM 功能，为普通用户的使用提供了更好的便利性和可控性。

(2) 利用可信计算思想，实现对 UEFI BIOS 的安全增强

虽然 UEFI BIOS 能够提供许多传统 BIOS 无法比拟的优点，如：结构层次清晰、由 C 语言开发、模块化并且易于实现等等。但是也有其缺点，如：UEFI 为网络安全堆积了巨大的安全隐患；UEFI 良好的扩展性和源码半公开的特点，使得黑客可以轻易找出 UEFI 的漏洞所在；UEFI 采用高级 C 语言进行编程开发，同时提供了 shell 调试接口，这也为攻击者

提供了便利，可能会造成对 UEFI 管理模式的攻击等。

本课题根据 UEFI 执行流程以及执行组件的特点，尤其是 UEFI 执行过程的核心阶段及其安全需求。从整体上设计基于 UEFI 的可信引导模型，构建了链式的 UEFI BIOS 信任链流程以增强 UEFI 的安全性。通过引入完整性验证，身份验证以及数字签名等功能，设计 UEFI 核心阶段的安全策略，并遵循 UEFI 标准进行实现。由此，确保 UEFI 核心阶段的各个组件在启动过程中不被篡改，并且保证 UEFI 核心阶段的各个组件能够以正确的顺序执行以及其所初始化的硬件的行为是可预期的；同时，确保用户的合法性，防止非法用户的入侵，提升终端的安全性。

（3）以 USB Key 为可信根，确保 Windows 系统引导过程的安全

操作系统引导过程是计算机系统的关键一环，如果该过程出现问题，那么整个系统的运行过程将处于一个不可信的状态，可能会给用户造成巨大的损失。

本课题通过分析 Windows 操作系统的启动过程以及其自身的保护机制，发现其可能存在的安全威胁。通过利用 USB Key 作为可信硬件对操作系统启动过程进行完整性度量和验证，确保操作系统启动过程中的各个组件的完备性和可靠性，大大提高 Windows 内核启动过程的安全性和可信赖性。

（3）为其他如基于虚拟机或者内核的安全机制提供最初的可信环境

本课题通过利用可信硬件 USB Key 实现系统终端的可信启动过程，在保证系统启动过程的安全可信的同时，也为其他的安全机制，如：内核级的可信操作，入侵检测，软件防篡改等安全技术，提供了最初的可信环境和基础，这些安全机制大多忽略了系统启动过程的防护，而系统的启动过程的可信是整个系统安全运行的基础，启动过程如果出现问题，则系统运行时的安全检测和防护也就失去了意义。所以，通过本课题的研究可以有效的提升其他安全机制的可信性和安全性。

（4）为其他终端类型的启动安全提供技术参考

目前，恶意行为针对的设备群体不断扩大。其他类型的终端设备，如手机、平板等移动终端，也在遭受着越来越多的恶意威胁^[1]。因此，如何确保该类设备的安全正在成为研究热点。本课题中利用的可信硬件 USB Key 实际上就是一个带有 USB 接口的智能卡，所以，利用本课题的思路，将智能卡应用到移动终端设备中可以实现可信启动过程。

（5）为其他 UEFI 下驱动程序的开发过程提供经验

通过深入分析研究 UEFI 规范及其 UEFI 框架实现过程，重点研究 UEFI 驱动技术，通过对智能卡和 USB 体系结构的研究，改进完善 UEFI 对各种 USB 设备的技术支持，并在 UEFI 系统架构中实现 USB Key 驱动程序以及应用程序，以实现 UEFI 对 USBKEY 的支持和实用。通过在 UEFI 下实现 USBKey 驱动程序，可以为 UEFI 下的其他驱动程序的开发提供经验。

1.3 国内外研究现状

1.3.1 可信计算研究现状

1999 年 10 月, 由 Intel, IBM 和 Microsoft 等公司一起成立了“可信计算平台联盟 TCPA (Trusted Computing Platform Alliance)”。TCPA 后来改组为可信计算组织 TCG (Trusted Computing Group), 进一步的增强了对系统安全的关注。TCG 制定并发布了可信硬件 TPM (Trusted Platform Module) 标准, 并提出可信计算思想。其基本思想可以概括为: “可信验证, 信任传递”, 主要过程是首先在计算机系统中建立一个可信根, 信任根的可信性由物理安全和管理安全来共同保证, 然后建立一条信任链。从计算机终端启动开始, 由信任根首先确保硬件平台的安全, 再由硬件平台确保操作系统的可信, 最后是系统的应用程序以及应用软件, 一级认证一级, 一级信任一级, 直到把这种信任扩展到整个终端系统, 从而保证了整个计算机终端系统的可信性和安全性。其信任链传递过程如图 1 所示。

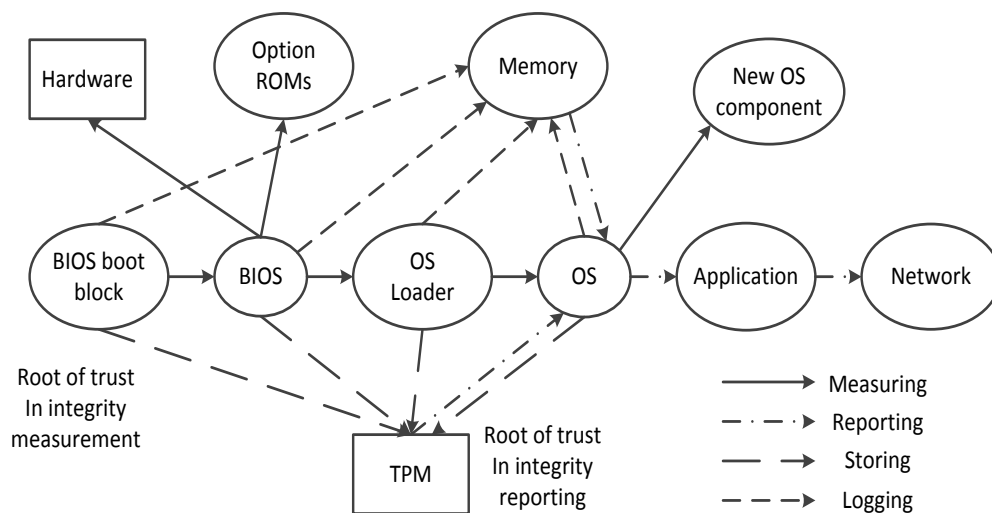


图 1 信任链流程

由图可知, 信任链以 BIOS boot block 和 TPM 芯片为根, 从 BIOS boot block 出发, 依次经过 BIOS、OS loader、OS, 最后到 Application, 一级度量一级, 一级信任一级, 保障了整个计算机启动过程的可信性和可靠性。但是根据可信计算理论: 信任在传递过程中可能会产生一定的信任损耗, 信任链传递的阶段越多, 其信任损失就可能越大。为此, 赵波, 张焕国等人提出了一种星形的信任模型^[16], 如图 2 所示。

相比于传统的链式过程, 这种星形的信任结构中, 任何一个被度量组件到信任根之间的距离都是一个单位。因此, 其信任传播的路径都达到最小, 其信任损失在信任传播的过程也达到了最小。

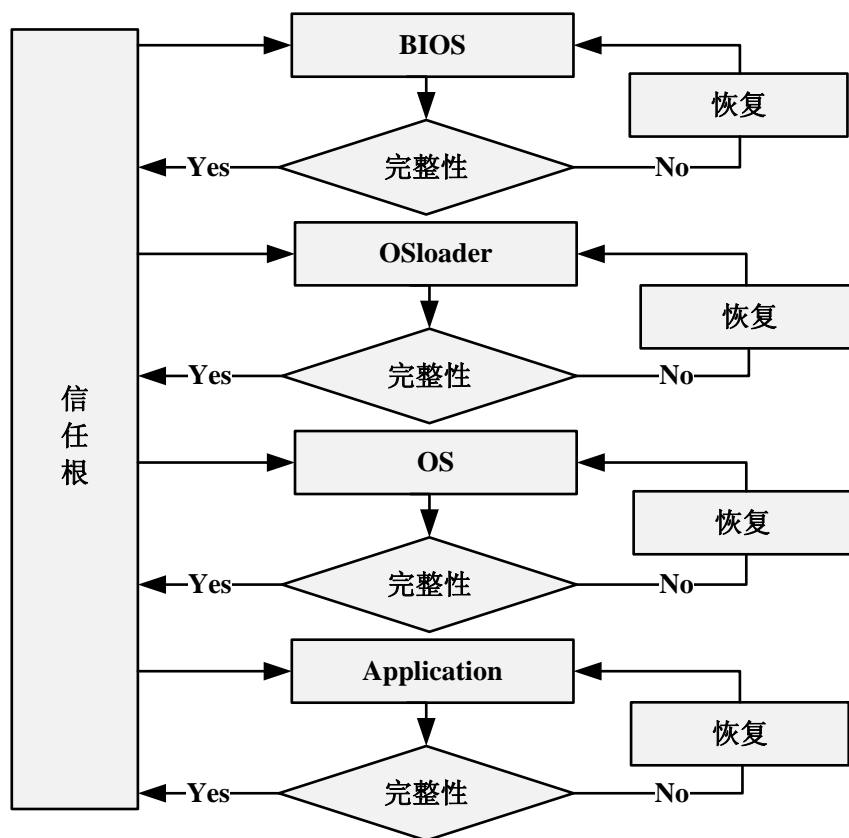


图 2 星形信任结构

可信计算已经成为当前安全领域研究的热点之一，各种产品也不断涌现。

在国内，2004 年，联想集团宣布开始着手于计算机可信芯片的开发，并在下半年开始对安全芯片进行测试和生产。在同一年十月，武汉瑞达信息安全产业股份有限公司研制成功了 SQY 14 嵌入密码型计算机，并通过相应的技术鉴定。这款嵌入密码型的计算机成为了国内第一款自主研发的可信计算平台。

在国外，早在 2002 年 IBM 就发布了基于可信计算技术的计算机。在之后一年，Intel 推出了 LaGrande 技术，该技术通过在计算机终端系统中构建一个可信子系统，通过该子系统可以实现防范恶意攻击以及保护计算机数据机密性等相关功能。微软也提出了 Palladium 可信计算计划（后来更名为 NGSCB——新一代安全计算平台），该计划利用软件和硬件提高 PC 的安全性，并已经应用于 Windows Vista 系统。

但是，可信计算也有一定的缺陷^[25]：

- （1）度量模型尚不完善，信任链过程的实现还有缺陷。
- （2）配套标准规范欠缺，度量技术的评判标准还有待发展。
- （3）目前的可信技术灵活性和适用性有待进一步的提高。

1.3.2 安全启动研究现状

目前，主要实现系统的安全启动过程主要有以下三种方式：基于软件的安全启动过程

基于可信硬件的可信启动过程以及利用虚拟机作为监控器来确保启动过程的安全性。下面，分别对这三种安全启动方式的研究现状展开论述。

1、基于软件的安全启动过程

基于软件的安全启动过程主要是利用恶意代码检测或者在系统中添加度量模块等方法来实现。如文献[22]论述了一种不依赖于 TPM 的可信引导度量过程，其方法主要是在操作系统启动之前先启动一个受保护的、用于验证操作系统运行环境的小型 Linux 系统，通过该系统验证 BIOS 和操作系统的启动过程的可信性和完整性，最后利用 Kexec 工具进行转换并启动用户磁盘操作系统。文献[31][38]通过将恶意代码检测的相关技术，如恶意行为检测、模式匹配以及样本检测等引用到 UEFI 系统中，为系统固件和操作系统的提供了安全防护功能。文献[27]通过建立 UEFI 平台下的恶意行为特征库的攻击树模型，设计了一种基于 UEFI 的最小攻击树模型检测方法。

2、利用可信硬件(包括 TPM、USB Key 和 PCI 卡等)实现系统的启动过程^{[17][24][26][28][34]}

在可信计算思想中，系统必须存在一个可信的原点，也叫可信锚点，并且该原点一般由可信硬件来担任，目前，可信计算思想得到了广泛的认可，国内外都进行了大量的研究。

在国内，文献[17]提出一种可配置的可信引导系统，其方法是用 Lois Grub 替换原有的 Grub 代码，实现的功能主要包括文件完整性的度量和验证、启动过程的顺序验证等，且在引导过程中，具有可配置的特点。文献[18]讨论了基于 TPM 构建系统信任链的来确保系统启动过程可信的可行性，文中通过在 MBR (Master Boot Record) 之前加入一个动态检测模块来完成信任链的传递过程。文献[47]阐述了一种利用通道技术，在主机和 TPM 之间采用并行工作方式，并支持被验证组件的备份和恢复。文献[37]提出了一种利用 PCI 卡作为可信硬件，并利用可信计算思想来实现系统的可信启动过程。

在国际上，Danmouth 大学开发了 Enforcer 安全模块^[24]，通过该安全模块修改 Linux 加载器 LILO (Linux Loder)，实现对 Linux 系统的安全防护，并利用 TPM 模块实现对文件系统的加密保护。在系统启动过程中，一旦文件的验证不通过，那么 TPM 将不会释放密钥，并且立即停止终端系统的启动过程；德国德累斯顿大学的 Bemhard Kauer 通过研究 L4 操作系统并利用 TPM 模块实现系统的认证启动工作^[34]；IBM 开发的 TCG Grub^[28]和德国波鸿大学开发的 Trusted Grub^[26]功能基本一致，它们遵循 TCG 的相关规范，主要思想是通过 TPM 模块实现对 Grub 系统的启动过程的完整性度量和验证工作。

3、利用虚拟机作为监控器实现系统启动过程的完整性度量

文献[13]提出利用虚拟机实现对操作系统引导过程的监控，确保系统启动过程的可信。

由此可见，目前关于终端启动的安全防护技术主要分为两个方面：一是基于软件的安全启动技术。这类安全启动技术主要是插入安全模块来保证系统启动的安全。二是基于可信硬件的可信启动技术，其方法主要是通过引入可信硬件 TPM 来确保系统启动的可信性和可靠性。相对而言，引入可信硬件的可信启动技术其安全性和可靠性要高于基于软件的

可信启动技术。

但是, TPM 芯片是嵌入在计算机主板上的, 用户如果要使用这种安全启动技术, 就势必要更换终端的主板。为此, 文献[48]提出利用 USB Key 代替 TPM 行使度量和验证操作, 文中分析了 USB Key 代替 TPM 模块的可行性, 并对其相关命令的效率进行了分析。基于此, 文献[50]提出了一种基于 USB Key 的可信启动方法, 主要思路是: 利用 USB Key 代替 TPM 行使可信根的相关功能, 但是, 本文只实现了对 BIOS 之后的启动组件的完整性度量, 并没有对 BIOS 进行安全防护, 也就是说它并不能防护 Bootkit 之类的攻击。针对这种情况, 文献[49]设计了一种在 EFI 环境下, 利用 USB Key 实现可信启动的方案, 该方案在 EFI BIOS 中添加驱动程序以实现 USB Key 的支持, 之后, 通过 USB Key 实现对 EFI BIOS 的安全增强。但是该文只是实现了对 EFI BIOS 的安全增强, 并没有实现对操作系统引导阶段的安全防护, 也没有实现在 EFI BIOS 度量出现问题情况下的后续措施, 系统的可用性受到很大的影响。

1.4 论文研究内容

本文以系统启动过程为研究对象, 重点关注可信启动的关键技术研究。在研究了当前可信启动研究现状的基础上, 着重分析了可信根 TPM 的优势与缺陷。由此, 在 UEFI BIOS 中引入 USB Key 设备, 设计了基于 USB Key 的可信启动框架。在遵循 UEFI 规范的基础上实现 UEFI 环境下的 USB Key 驱动程序和应用程序, 进而以混合信任链的方式实现该终端系统的可信启动。最后通过实验对本文实现的可信启动系统功能和性能进行了测试与分析。

研究工作主要有以下几点:

1、UEFI 规范及其脆弱性研究

近年来, UEFI 的相关研究已经取得了长足的发展, UEFI 替代传统的 BIOS 已经成为了一种趋势, 但是, 其扩展性在给用户带来便利的同时, 也给系统带来了潜在的不确定性和威胁。本文分析研究了 UEFI 的框架、关键组件及其驱动模式的基础上, 重点研究其运行机制, 并分析 UEFI 在运行过程中的脆弱性, 为可信启动过程设计与实现提供相关技术和理论支持。

2、基于 USB Key 可信启动架构的设计

通过对可信平台模块 (TPM) 的优劣性进行分析, 本文引入了更加灵活的可信硬件 USB Key, 并依据 UEFI 阶段以及操作系统引导阶段脆弱性设计了基于 USB Key 的可信启动系统架构。由于信任在传递过程中会出现一定的信任损耗, 并且传递路径越长, 损耗越大。而相比链式的信任链过程, 星形的信任链结构在传递过程中信任损耗最小。由此, 根据 UEFI BIOS 阶段和操作系统引导阶段的独特性, 提出本架构采用混合信任链的信任传递过程。该过程在 UEFI 启动过程中采用链式的传输过程, 而在操作系统引导阶段采用星型结构的链式传输过程。并且, 为了确保系统的完整性和可用性, 在系统中添加了恢复模块,

以实现在发现系统遭受到恶意攻击时，能够对系统进行及时恢复。

3、UEFI 下 USB Key 驱动程序与应用程序的设计与实现

UEFI 在新一版的规范中，虽然提出了 CCID（USB Chip/Smart Card Interface Devices，USB 芯片智能卡接口设备）驱动程序，但是它并没有为 USB Key 提供明确的命令接口，也没有提供相应的源码。为此，为了实现在 UEFI BIOS 环境下对 USB Key 设备的控制和访问，需要设计和实现 UEFI 下的 USB Key 驱动程序。本文通过对 UEFI 驱动驱动模式、UEFI 中 USB 驱动协议栈以及 USB 设备系统结构进行分析研究，设计并实现了 USB Key 的驱动程序，实现了在 UEFI 环境下对 USB Key 设备的访问与控制。为实现具体的命令操作，本文根据 USB Key 的命令系统，设计并编码实现了在可信启动架构中所用到的应用程序。并利用 UEFI 中已有的安全模块和协议，提高了系统的效率。

4、建立基于白名单机制的基准库以及基于 UEFI Capsule 机制的恢复模块

基准库和恢复模块是可信启动系统的重要组成部分，它们的安全性和可靠性关系到整个可信度量系统的结果与响应是否安全可靠。同时，也需要考虑到的是硬件在系统启动过程中的关键作用。现有的可信启动系统都只是考虑了系统中的可执行文件的安全性，并没有对硬件作出相关限制，用户有意或无意的增加或删除部分硬件，可能会给系统带来巨大的影响。本文首先分析了目前存储基准值的两种主流方法，在综合考虑安全与效率的情况下，选择了将系统在完整性度量过程中所需的基准值建立在主机端，并通过 USB Key 签名保护。其次，建立硬件白名单，系统在发现新硬件后，首先检测其是否在白名单中，如果在则加载驱动并初始化该硬件，如果不在，则提示管理员采取相应的措施。为避免系统过于臃肿，将硬件白名单与可执行文件的基准值合在一起，建立基准库文件 standard，并由 USB Key 确保该文件的安全性。

其次，为了确保系统在出现问题时，能够产生可靠的安全响应。文中提出了基于 UEFI Capsule 机制的恢复模块。在管理员模式下，可以实现对该模块的调用，降低系统因遭受恶意攻击而造成的损失。

5、基于 USB Key 可信启动系统的实现与测试

依据设计基于 USB Key 的可信启动架构以及已经实现的 USB Key 驱动程序和应用程序，就可以实现基于 USB Key 的可信启动系统，系统采取混合信任链流程。系统实现对每一个将要运行的组件进行完整性度量与验证，只有在通过验证后系统才能执行该组件。同时，系统也对用户的身份进行合法性认证和级别认证，避免非法用户登录系统而造成损失。最后，对该系统的功能和性能进行测试，验证本文所设计系统的正确性和有效性。

1.5 论文结构安排

论文正文分为六章，各个章节内容安排如下：

第一章介绍了课题背景和国内外关于可信计算与可信启动的研究现状，阐述了课题研

究的意义，并给出了本文的主要研究内容。

第二章首先对可信计算理论进行了简要介绍，并对 TPM 模块进行了分析，阐述了 TPM 的优势与缺陷。随后介绍了 UEFI 及其框架，分析了 UEFI 的关键组件，包括启动管理器、核心固件和 UEFI 映像。最后对 UEFI 的驱动模式以及其运行机制进行了分析研究，为基于 USB Key 的可信启动技术的研究奠定了基础。

第三章分析并设计了基于 USB Key 的可信启动系统框架。首先给出了可信环境下的信任链模型。之后，通过对当前安全需求的分析，从设计原则、安全目标以及设计思路出发，设计了基于 USB Key 的可信启动系统框架，并给出了基于混合信任链的可信启动流程。最后，提出了该系统在实现过程中所涉及的关键技术。

第四章设计并实现了 UEFI 下 USB Key 的驱动程序。首先分析研究了 USB 的系统结构以及 UEFI 下的 USB 驱动程序栈。然后依据 UEFI 驱动模式，给出 USB Key 驱动程序所涉及的关键函数，并详细阐述了其实现流程，并编程优化了 USB Key 的驱动程序。

第五章设计并实现了 UEFI 下 USB Key 的应用程序。驱动程序仅能够实现对 USB Key 设备的可识别、可访问，并不能实现对 USB Key 的具体命令操作，为此，本章首先研究了 USB Key 的命令系统，分析其命令应答机制、指令规范以及命令处理流程。据此，利用 UEFI 的相关协议和 USB Key 驱动程序实现 UEFI 下的 USB Key 应用程序。

第六章实现了基于 USB Key 的可信启动系统，并对其进行测试分析。首先设计并实现了系统架构中的辅助模块，建立了基于白名单机制的基准库和基于 UEFI Capsule 机制的恢复模块。然后将第四章和第五章所实现的 UEFI 下的基于 USB Key 驱动程序、度量程序和度量程序整合到可信启动系统中，使之成为一个完善、易用的基于 USB Key 可信启动系统。最后对该系统进行了详细的功能和性能测试，并与其他主流方案进行对比分析。

最后是对本文研究内容的总结和评价，并对未来可能的研究工作进行了讨论。

第二章 论文相关理论基础

通过 USB Key 设备实现系统的可信启动过程,可以更加灵活高效的保证系统启动过程的可信性与可靠性,并且可以为其他的如基于内核或者虚拟机的系统安全机制提供最初的可信环境。为了设计和实现基于 USB Key 的可信启动系统,本章首先介绍了可信计算理论的相关基础,并简要分析了可信平台模块(TPM),随后讨论了 TPM 与 USB Key 各自的优势与缺陷。之后对 UEFI BIOS 的框架、关键组件以及运行机制进行了简要的介绍,为实现可信启动系统提供理论支撑。

2.1 可信计算概述

2.1.1 可信计算简介

众所周知,计算机底层硬件和在其上运行的操作系统的可靠与安全是整个系统安全的最基础也是最关键的一环,若要从根本上确保整个计算机系统运行环境的安全,就必须从最底层做起。为了全面保护计算机终端的安全,必须考虑各方面的影响因素并采取措施,这就需要从 BIOS、底层固件和硬件以及操作系统等方面进行综合保护。正是基于这种思想,促使了可信计算理论的诞生。可信计算的概念最早产生于二十世纪八十年代,并在 1999 年,微软、IBM 等著名 IT 企业巨头联合成立了可信计算平台联盟(TCPA),标志着可信计算理论的蓬勃兴起。之后,在 2003 年可信计算平台联盟正式改组为可信计算组织(TCG)^[42],可信计算的发展迎来新一轮高潮,可信计算理论及其应用领域得到进一步发展和扩大。

广义来说,可信计算理论是通过在计算机中引入安全硬件模块,以提高整个计算机系统的安全性。其基本思想是在计算机系统中构造一个物理安全信任根,并以此信任根为起始构建一条从系统加电启动到应用加载的完整信任链,信任链中每一个节点度量和验证后一个节点,直到最后将信任范围扩展到整个计算机系统,使系统启动过程中的每一步都是经过验证的,是可信的,从而确保计算机运行环境的可信性和可靠性。随着当前信息系统安全形势的日趋复杂,可信计算已经成为信息安全领域的研究热点之一,并已经取得了令人鼓舞的成就。

2.1.2 可信平台模块分析

TCG 提供了可信平台模块 TPM^[21],通过 TPM 可信实现完整性度量和验证、数据的加密与解密等相关功能。TPM 是一个被动的存储设备,它以硬件方式嵌入到主板上。在 TCG 规范中,TPM 是可信计算平台的信任根,是可信链的可信原点。TPM 的内部结构如图 3 所示,它主要由存储器、密码相关部件、I/O 接口和执行引擎等 11 个部件构成组成。

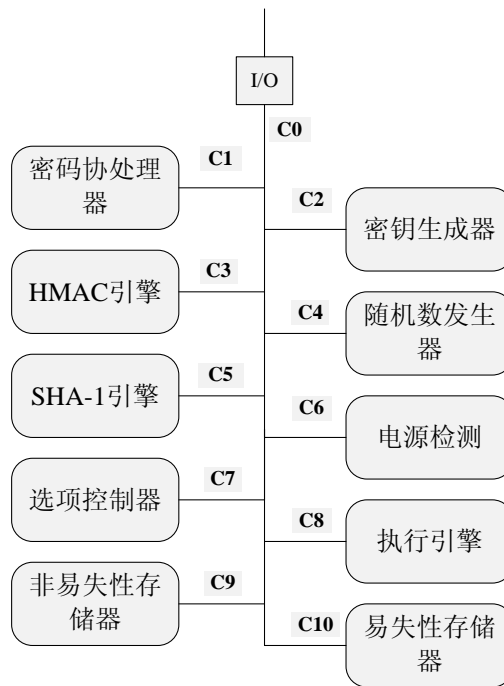


图 3 TPM 内部结构示意图

其中 I/O 接口主要负责将接收到的数据和指令传送到其他相应部件，也就是说 I/O 接口可以实现对 TPM 通信总线上的信息流进行分流管理；密码协处理器负责协助处理 TPM 内部的密码相关操作；随机数产生器主要利用单向散列函数将一个未知长度的输入数据转换成一个定长为 32 字节的随机数，之后将该随机数提供给 TPM，并且该随机数只能用于 TPM 内部，不能被取出；密钥生成器则可以利用随机数产生器产生的随机数生成 RSA 非对称密钥对；HMAC（Hash-based Message Authentication Code）引擎通过向 TPM 提供认证码来确保 TPM 内部的数据和命令的完整性；SHA-1 引擎用于完成 TPM 中关于 hash 的运算操作，以便完成完整性度量和验证过程；执行引擎根据 TPM 命令执行相关操作；电源监测模块则主要是通过检测平台的电源状态来管理 TPM 的电源状态，以帮助 TPM 在电源状态发生变化时采取适当应对措施；非易失性存储器主要用于存储永久性标识，其主要目的是保护 RSA 密钥对的安全；易失性存储器主要负责存储临时性数据和标识，主要包含平台配置寄存器和身份验证密钥等，一旦系统遭遇到断电等突发情况则存储在其内部的数据将全部丢失。

虽然 TPM 为 TCG 组织指定的可信根芯片，但是由于 TPM 芯片是内嵌在主板上的，所以普通用户不易实现对其的更换或者扩展。再者，对于许多单位来说，并不是所有的计算机都配有 TPM 芯片，如果想要使用 TPM 芯片，就需要更换主板或者更换主机。同时，为了避免潜在的威胁和不确定性，管理中心更希望对用户进行分等级管理，例如普通用户只能使用该计算机但不能对计算机的安全系统进行更改；而管理员不仅可以使计算机也可以对终端的安全模块进行扩展和更新。但 TPM 芯片并不能很好地支持这种分级操作。

2.2 统一可扩展接口

2.2.1 UEFI 简介

随着计算机的快速发展，传统的 BIOS 逐渐成为了制约计算机发展的瓶颈。每当在计算机终端需要添加新的硬件或者新的平台功能时，都要求 BIOS 的相关开发人员对 BIOS 进行日趋复杂的设计和更改。与此同时，操作系统也需要作出相应的配合和更改。这个过程不仅耗时耗力，也会浪费很多资源。由此，由 Intel, Microsoft, AMI 等巨头企业联合组成了统一可扩展固件论坛（United Extensible Firmware Interface Form, UEFI Form）。

UEFI 提供了一个统一的，可扩展的平台。它将平台的硬件和固件抽象成为了一系列标准的接口，利用这些接口，操作系统只需要和平台固件传递必要的信息就可完成系统的引导过程，而并不涉及平台的具体细节。这样，就便于用户或厂商可以在不影响操作系统启动过程的情况下，通过开发驱动程序或者应用程序为平台增加新的功能部件或者硬件设备。

UEFI 的设计主要基于以下三点：

1. 兼容已有的接口标准

UEFI BIOS 为了实现对当前现有的固件和操作系统的支持，而兼容了当前已有的接口标准，例如 ACPI（Advanced Configuration and Power Management Interface）接口标准和 SMBIOS（System Management BIOS）接口标准等。

2. UEFI 系统分区

不同 BIOS 厂商可以安全地共享 UEFI 系统分区，该分区建立在大容量的存储介质上。通过对该分区的共享很好的解决了由于系统不断增加新功能而导致非易失性存储器容量持续扩大的问题。在其中可以存放符合 UEFI 规范的驱动程序及应用程序等可执行代码。同样，UEFI OS loader（操作系统引导程序）也可以作为一种特殊的应用程序存放在该分区内。目前，UEFI 系统分区只支持 FAT32 文件系统。

UEFI 系统分区的存在，使得 UEFI BIOS 的驱动程序或者应用程序可以存放在其他的非 BIOS ROM 的存储介质中，这就增加了 UEFI BIOS 程序存储的灵活性，极大地促进了 UEFI BIOS 的扩展性。

3. 运行时服务和引导时服务

引导时服务（Boot Service）主要负责在引导过程中调用 UEFI BIOS 接口和访问可用设备提供服务。其中，通过 UEFI 规范中的“句柄（Handle）”和“协议（PROTOCOL）”可以实现对可用设备的访问，这有利于对现有 BIOS 程序的重用，减少重新开发的工作量。运行时服务（Runtime Service）则负责对硬件平台资源做一定的抽象，便于操作系统在运行过程中的访问。

图 4 展示了 UEFI 的各个关键组件之间的逻辑关系^[44]。

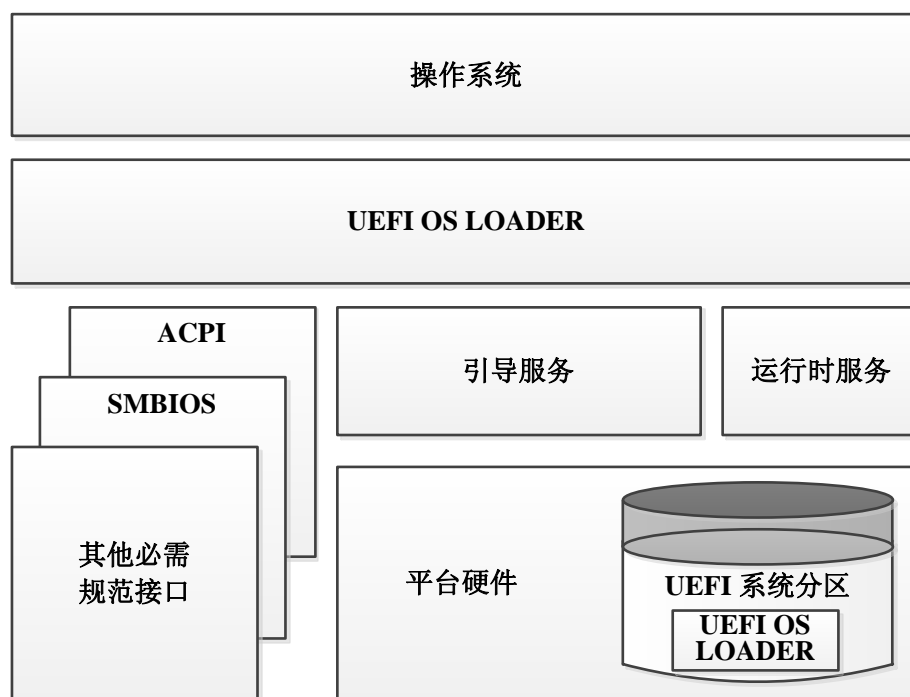


图 4 UEFI 逻辑结构示意图

由图可知，在支持 UEFI 的系统中，通过各个关键组件之间相互作用实现终端系统的引导过程。UEFI 可以在启动过程中从 UEFI 系统分区内找到用于加载的操作系统引导文件，并且很多大容量的存储设备也可以作为 UEFI 系统分区，例如 DVD、磁盘、CD-ROM 甚至可以是需要借助网络传输的远程启动设备等。此外，如果考虑可以对 OS loader 做适当修改，那么，借助 UEFI 的可扩展功能，可用于引导操作系统的存储媒介类型还可以进一步增加。在 OS loader 运行之后，便开始操作系统引导的相关工作。在这期间，OS loader 可能还需要用到 UEFI 相关的服务或接口，以实现对各个平台组件和上层的操作系统软件的检测、识别以及初始化。需要注意的是，UEFI 运行时服务也可能会在引导过程中为 OS loader 提供一些服务。

2.2.1 UEFI BIOS 框架

UEFI 仅仅是一个规范，只定义了平台的一些接口标准，还不能够直接运用到实际系统中。为此，UEFI 组织又提出了 UEFI 的 PI (Platform Initialization, 平台初始化) 规范^[45]，为 UEFI BIOS 的具体实现过程提供了依据。依据 UEFI 规范与 PI 规范，BIOS 生产厂商可以结合自身需求实现具体的 UEFI BIOS。UEFI BIOS 系统框架如图 5 所示。

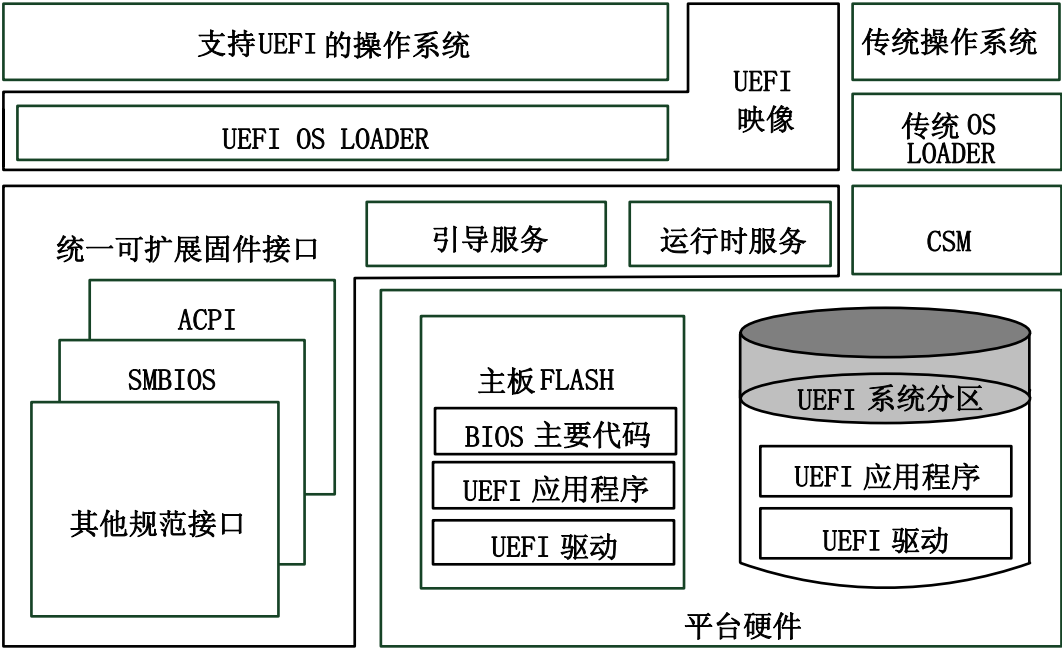


图 5 UEFI BIOS 系统框架

图 5 展示了 UEFI BIOS 的系统框架，整个系统包含了平台硬件、可扩展借口、UEFI 相关映像、操作系统以及为了兼容非 UEFI 操作系统的 CSM(Compatibility Support Module) 模块。

一般的 UEFI BIOS 主要包含以下几个部分：UEFI 启动管理器，UEFI 系统表，句柄服务库，UEFI 核心固件以及 UEFI 映像文件等。其中，UEFI 核心固件包含 EFI 服务和 EFI 协议两个方面；UEFI 映像文件主要包含 EFI 应用程序和 EFI 驱动程序两种可执行文件。本课题重点关注关键组件中的启动管理器，核心固件以及 UEFI 映像文件三个方面。

2.3 UEFI 关键组件

2.3.1 启动管理器

启动管理器可以通过 UEFI 映像加载服务，在符合 UEFI 系统的文件系统中加载 UEFI 驱动程序或 UEFI 应用程序(包括 UEFI OS loader)。UEFI 定义了一些 NVRAM(Non-Volatile Random Access Memory) 变量，通过这些变量 UEFI 可以维护一个引导菜单。启动管理器根据 NVRAM 变量就可以按照一定的顺序加载 UEFI 驱动程序或者 UEFI 应用程序，所以通过对引导菜单中的 NVRAM 变量的进行增加或者删除操作,就可以控制 UEFI BIOS 的加载流程。但是，需要说明的是引导信息是由上层软件对 NVRAM 进行设置来确定的，UEFI BIOS 本身并不能获得启动管理器的引导信息。同时，通过这些引导信息也可以明确 OS loader 与操作系统内核之间的相互位置关系。

另外还需要说明的是，UEFI BIOS 在启动管理器中增加一些额外功能以应对在系统引

导过程中可能发生的突发情况。例如，在系统引导过程中如果发生严重的错误，那么启动管理器将会启动一个原始设备制造商（OEM）自带的诊断检测工具，用以检测自身的运行环境问题。

2.3.2 UEFI 核心固件

1、UEFI 服务

UEFI 服务本质上是一系列的接口函数，它为 UEFI 在加载映像文件时，提供了统一的、通用的启动环境。UEFI 服务主要包括启动时服务（Boot Service）和运行时服务（Runtime Service）。定义这些接口函数可以使平台和操作系统之间能够相互独立。同时，运行时服务也为操作系统运行阶段提供了一组最小单元的调用接口。而且，这些服务使用 64 位接口，为以后的扩展留下了空间。

1) 引导服务（Boot Service）

在引导过程中，UEFI BIOS 通过引导服务控制着系统资源。针对不同对象，引导服务接口函数可以分为基于句柄（Handle-based）和全局（Global）两类。其中基于句柄的函数只能访问特定的设备或者设备功能，而全局函数则可以在所有的平台上访问系统服务。所有的 UEFI 应用程序（包含 UEFI OS loader）必须使用引导服务接口函数来访问设备以及分配内存。在 UEFI OS loader 调用 ExitBootServices（）函数后，所有的引导服务都将终止服务，而运行时服务则能够继续工作。引导服务接口主要分为以下四类：

- 设备协议
- 协议服务
- 全局引导服务接口
- 基于句柄的设备引导服务接口

2) 运行时服务（Runtime Service）

运行时服务是这样的一系列接口函数，它可以将平台硬件经过抽象后为操作系统提供所需的服务。运行时服务在 UEFI BIOS 运行结束后不会释放，因此，它可以同时在系统引导过程和操作系统运行过程中提供系统所需的服务。分配给运行时服务的内存空间始终会被保留，任何情况下其他组件都不能占用该空间。运行时服务所使用的硬件资源由 UEFI BIOS 确定，同时，操作系统也会在调用运行时服务时获得其在内存中的相关空间信息，并确保该部分内存不会分配给其他的模块。运行时服务主要分为以下三类：

- 变量服务
- 时间服务
- 虚拟内存服务

2、UEFI 协议

UEFI 协议与 C++语法中的类相类似，它将协议的服务和数据成员作为成员函数和数据成员进行封装。UEFI 核心固件为了便于管理 UEFI 协议，建立了一个 UEFI 协议库

(Protocol Database)。设备句柄所关联的协议都可以通过句柄协议 `HandleProtocol()` 或者打开协议 `OpenProtocol()` 等引导服务在 UEFI 协议库中找到。协议的实际名称由 GUID (Global Unique ID) 来表示, 而一个 UEFI 协议中可能会有多个 GUID。此外, UEFI 协议还包含了协议的接口数据结构和协议服务。通常情况下, 运行时的内存空间不可以分配给协议的接口结构, 操作系统运行时也不能调用协议的服务也就是其成员函数, 而且为了保证在多处理系统运行的安全性, 协议的成员函数不能被重复调用。

用户可以根据自身需求在已定义协议的基础上增加具有特殊功能的协议就可以实现对 UEFI BIOS 功能的扩展。所以, UEFI 协议的存在增强了 UEFI BIOS 的扩展性^[46]。

为了验证该协议是否支持某个设备, 可以通过引导服务中的 `HandleProtocol()` 或者 `OpenProtocol()` 检测此设备句柄是否支持相应协议的 GUID。如果该句柄支持这个协议, 那么将返回一个指针, 通过这个指针可以调用该协议定义的服务。图 6 为 UEFI 协议的组织结构示意图。通常情况下, 一个 UEFI 驱动程序可能会包含一个或者多个 UEFI 协议。

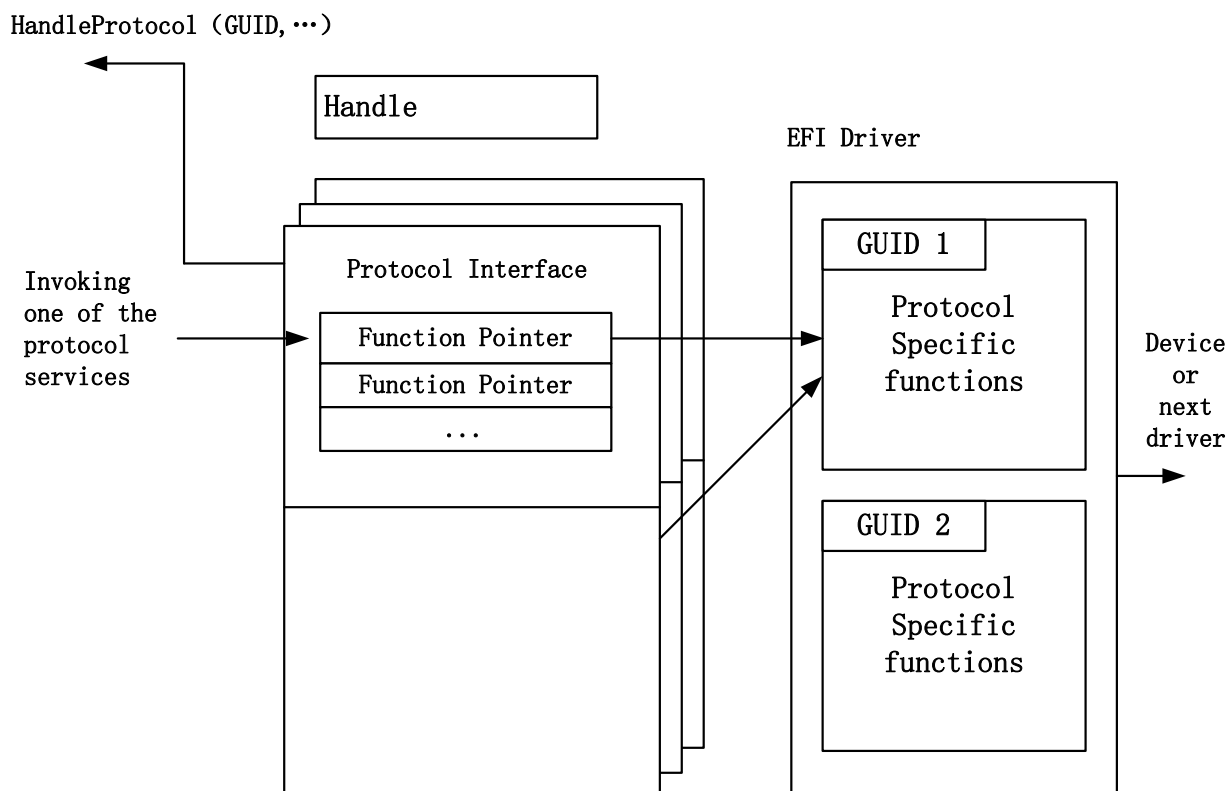


图 6 UEFI 协议

由图可知, 每个 UEFI 句柄对应着一个链表, 协议可以由 GUID 来表示, 协议接口内的功能指针则指向了协议实例, 它封装了数据成员和成员函数。

在协议数据库中, 协议是安装在句柄 (Handle) 上的。而若干句柄和协议组成了 UEFI BIOS 的核心组件: 句柄数据库, 如图 7 所示。其中, 句柄是一个数据结构, 代表了管理状态的临时值。每个句柄都具有一个独特的编号, 为访问句柄数据提供了入口。每个句柄

上包含有若干个协议。而又由上文可知，协议是一类由 GUID 命名的数据结构，在协议中可能包含某些数据或服务，也可能为空。在 UEFI BOIS 启动过程中，UEFI BIOS 会根据需要创建句柄，并在句柄上按照需求挂载合适的协议，同时也就确定了句柄的类型。由于句柄数据库是全局结构，所以在 UEFI 启动过程中所有的可执行文件都可以访问到句柄数据库。一般情况下，句柄能够表示 UEFI BIOS 组件主要有可执行映像文件、设备和 UEFI 服务，分别对应了句柄数据库中的 Agent 句柄、控制器句柄和服务句柄。

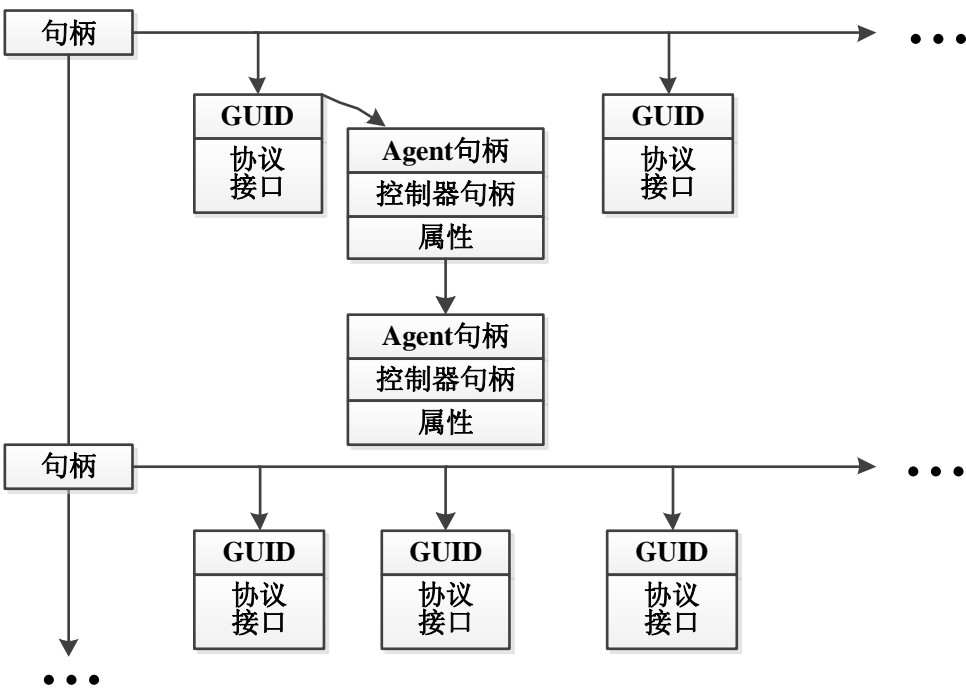


图 7 句柄数据库示意图

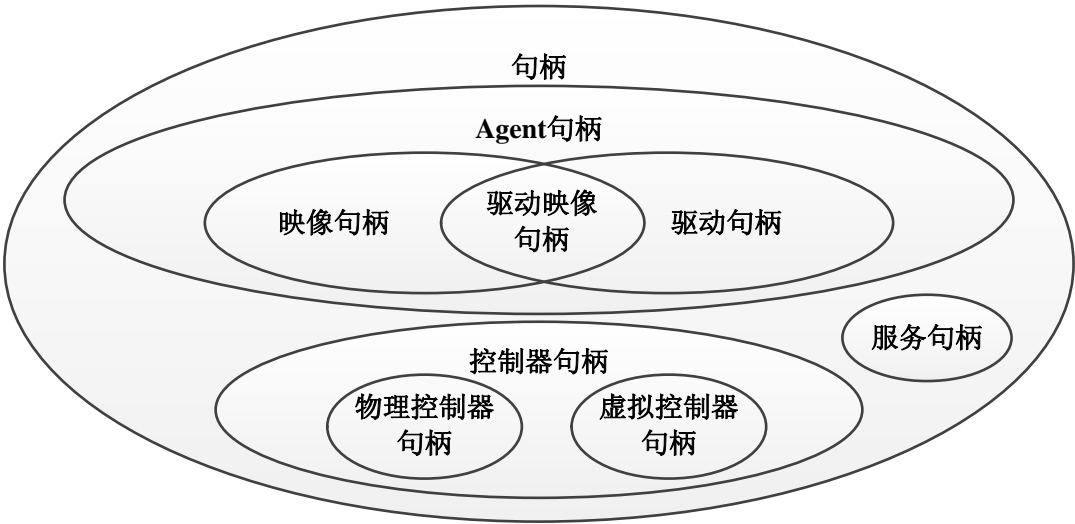


图 8 句柄类型及其间的逻辑关系

所有句柄的类型是由其上挂载的协议类型所决定的。句柄数据库中所有的句柄类型，以及句柄类型之间的逻辑关系如图 8 所示。句柄主要分为 Agent 句柄、句柄控制器和服务句柄三种，而 Agent 句柄包含了映像句柄和驱动句柄；控制器句柄则包含物理控制器句柄和虚拟控制器句柄。

2.3.3 UEFI 映像

UEFI 映像是一类由 UEFI 规范定义的可执行代码文件。UEFI 采用 PE32+映像格式，这种格式是对 Windows 的 PE/COFF 中的 PE32 的修订，其后面的“+”表示对普通 PE32 格式提供 64 位的重定位。PE32+最主要的特点是文件的开始部分是标准的文件头部，用以描述文件随后的可执行文件代码。系统可以通过 LoadImage（）引导时服务将 UEFI 映像文件加载到内存中，并将系统的控制权传递给相应的映像入口函数。UEFI 映像可以分为两类：UEFI 驱动程序和 UEFI 应用程序。其中 UEFI 驱动程序包括 UEFI 引导服务驱动和 UEFI 运行时驱动，如表 1 所示，其中 UEFI 映像文件头部的子系统值表示了 UEFI 映像的具体类型。UEFI 应用程序和 UEFI 驱动程序的区别主要在于系统将它们加载到不同的内存空间，而且二者在退出或者返回时的动作也有些区别，如应用程序在退出时，系统会释放该应用程序所占的内存；而驱动程序只有在返回出现错误时，系统才会将其内存卸载。

表 1 UEFI 映像类别

映像名称	映像文件头部子系统值
UEFI 应用程序	10
UEFI 引导服务驱动	11
UEFI 运行时驱动	12

根据 UEFI 规范及其可扩展功能，多种存储介质可以用于存储 UEFI 映像文件。目前，比较常用的 UEFI 映像存储介质主要有以下四种：

- PCI 卡上的扩展 ROM
- BIOS 芯片中的 Flash
- 媒体存储设备
- LAN（Local Area Network）引导服务器

下面分别对 UEFI 应用程序和驱动程序进行分析研究。

1、UEFI 应用程序

UEFI BIOS 可以通过启动管理器加载 UEFI 应用程序（UEFI Application），也可以由其他 UEFI 应用程序加载并执行。其加载过程如下：

在某个应用程序需要加载时，系统首先为应用程序分配足够大的内存空间，确保该内存空间可以容纳该应用程序映像文件。然后将应用程序中不同的段拷贝到内存的指定区域中，在拷贝完成后进行重定位操作，确定程序段或者函数之间的相对位置。在前面的所有

操作完成之后，系统会将该部分内存空间标识成为应用程序指定的类型，此时，应用程序的入口函数获得系统的控制权，加载过程结束。

当 UEFI 应用程序退出或者调用 `Exit()` 引导服务时，系统控制权将转回给加载它的 UEFI 组件，并从相应的内存空间中卸载该应用程序，它所占用的内存空间也会被回收。

UEFI OS loader 是一种特殊的 UEFI 应用程序，一旦 UEFI OS loader 加载，它会接管系统的控制权并负责加载操作系统。如果 UEFI OS loader 在加载或者执行过程中出现问题，它会将系统的控制权转回给 UEFI BIOS，并通过调用 `Exit()` 引导服务来释放已分配资源，提示操作系统加载失败。如果操作系统加载成功，它会通过调用启动服务的 `ExitBootServices()` 来终止所有的启动服务，正式接管系统。

2、UEFI 驱动程序

UEFI 驱动 (UEFI Driver) 可以通过启动管理器，UEFI 其他组件或者 UEFI 应用程序加载。与 UEFI 应用程序加载过程一样，在某个驱动程序需要加载时，系统首先为驱动程序分配足够大的内存空间，确保该内存空间可以容纳该驱动程序映像文件。然后将驱动程序中不同的段拷贝到内存的指定区域中，在拷贝完成后进行重定位操作，确定程序段或者函数之间的相对位置。在前面的所有操作完成之后，系统会将该部分内存空间标识成为与驱动程序相对应的类型，此时，驱动程序的入口函数获得系统的控制权，加载过程结束。

但是驱动程序的退出过程与 UEFI 应用程序并不一致，驱动程序只有在程序出错或者主动调用了 `Exit()` 引导时服务时，该 UEFI 驱动才会被卸载，并将控制权返回给调用者。

UEFI 驱动可以分为 UEFI 引导服务驱动 (UEFI Boot Driver) 和 UEFI 运行时驱动 (UEFI Runtime Driver)。二者主要区别在于在系统调用 `ExitBootServices()` 引导服务后，运行时驱动可以继续存在于系统中，而引导服务驱动在操作系统运行时不能被调用，其所占用的内存都将在引导结束时被回收。UEFI 运行时驱动通过调用 `SetVirtualAddressMap()` 运行时服务，可以将 UEFI 运行时驱动的物理地址转换到虚拟地址，从而可以在操作系统环境使用 UEFI 运行时驱动。

2.4 UEFI 驱动模式

UEFI 组织设计驱动模型的目的是简化驱动程序的设计和实现，并产生较小的可执行映像文件。根据 UEFI 驱动模型，其驱动程序需要包含 `DriverEntryPoint` 和 `EFI_DRIVER_BINDING_PROTOCOL` 两个部分。`DriverEntryPoint` 是驱动程序的入口函数，`EFI_DRIVER_BINDING_PROTOCOL` 是实现驱动程序的主要部分，包括 `supported()`、`start()` 和 `stop()` 三个函数，其中，`supported()` 函数用于检测驱动程序是否支持指定的硬件设备；`start()` 函数实现对硬件设备的初始化管理；`stop()` 函数用于终止驱动程序和硬件设备的连接。另外，为了便于管理驱动程序所对应的设备，UEFI 驱动模式定义了 `EFI_COMPONENT_NAME_PROTOCOL` 协议，利用该协议可以在 UEFI 环境中以自定义的

标示符表示该设备。驱动程序的初始化过程如图 9 所示。

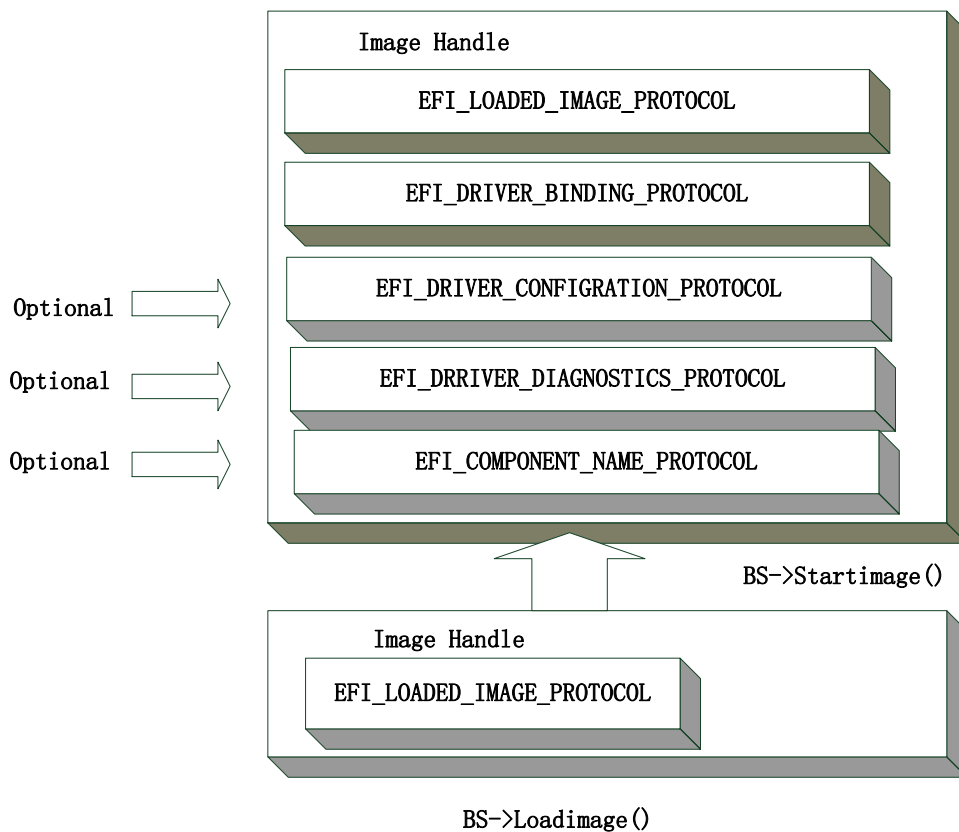


图 9 驱动程序的初始化过程^[43]

系统在找到驱动程序映像之后，首先利用 Boot Service 中的 LoadImage（）引导服务将映像文件加载到内存中，并为其分配一个映像句柄（Image Handle），同时在映像句柄上安装 EFI_LOAD_IMAGE_PROTOCOL 协议。随后系统将调用 StartImage（）引导服务开始驱动程序的初始化与执行过程。此时，系统将控制权交给驱动的入口函数 DriverEntryPoint，从而开始驱动程序的初始化。当驱动程序初始化成功之后，用户就可以利用 Boot Service 中的 ConnectController 服务使该驱动程序与该驱动程序对应的硬件设备相连接。ConnectController 服务可以得到一个驱动程序映像句柄表，并通过句柄上安装的 Supported（）函数来测试驱动程序是否支持该硬件设备。测试成功后，系统将控制权移交给 EFI_DRIVER_BINDING_PROTOCOL 实例，并利用实例中的 Start（）函数实现对设备的初始化和管理工作。利用实例中的 Stop（）函数，实现驱动程序与硬件设备断开连接的操作。

2.5 UEFI 运行机制

UEFI 的启动过程主要分为五个阶段：SEC（Security）阶段，PEI（Pre-EFI）阶段，DXE（Driver Execution Environment）阶段，BDS（Boot Device Select）阶段和 RT（Run Time）阶段。其启动过程如图 10 所示。

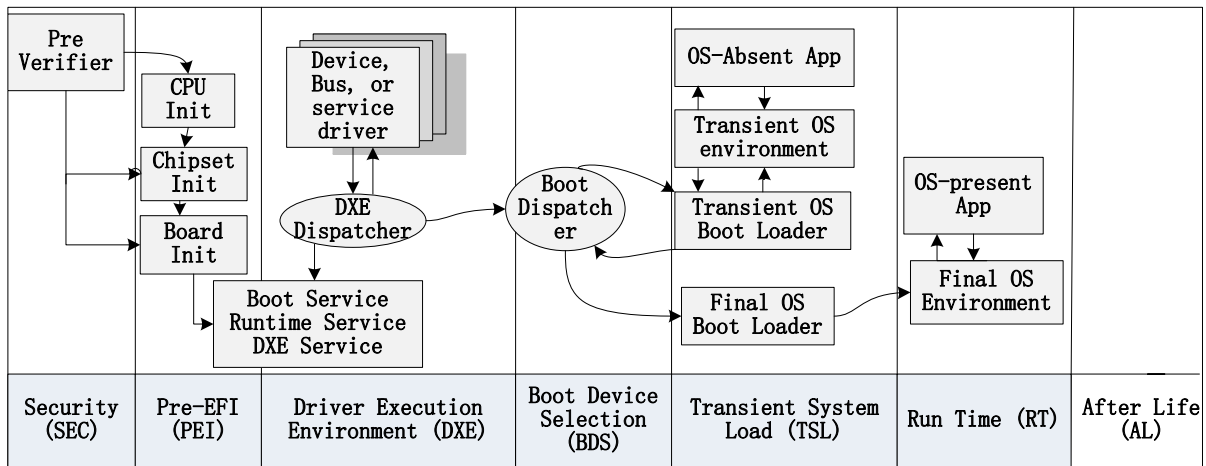


图 10 UEFI 运行过程

● SEC 阶段

Sec 阶段是计算机终端上电后的第一个阶段，它主要确保最先执行的程序是可以信赖的，并为 PEI 阶段的执行提供必要的支撑。

● PEI 阶段

在 PEI 阶段需要配置系统，以达到满足 DXE 阶段执行的最低的先决条件。在这一阶段，系统对主板上的一些芯片进行初始化，并建立起 C 语言的执行环境。PEI 阶段主要包含了 PEI Foundation 和 PEIMs (Pre-EFI Initialization Modules)。其中，PEI Foundation 位于 BFV (Boot Firmware Volume) 中，PEIMs 位于 FV (Firmware Volumes) 中。该阶段的主要工作是实现加载、验证和分发 PEIMs；PEIMs 之间的通信；向 DXE 阶段传递必要的数

● DXE 阶段

DXE 阶段主要由 DXE Foundation，DXE 分发器和一系列的驱动程序组成。其中，DXE Foundation 产生了一系列的 Boot Service，Runtime Service 和 DXE Service；DXE 分发器主要负责检测驱动并使驱动按照一定的顺序执行；驱动程序则实现了对处理器、芯片组和平台相关组件的初始化。驱动程序的加载顺序是由 BFV 中的启动顺序管理文件(a priori file)、驱动中的依赖表达式决定的。DXE 分发器首先找到启动顺序管理文件中列出的驱动程序，使其按顺序加载。之后，再寻找剩余的驱动程序，检测其依赖表达式，当其依赖表达式满足时，才能加载到内存中。

● BDS 阶段

BDS 阶段是 UEFI 启动的最后一个阶段，主要负责从可用的设备中启动用户选择的操作系统。同时，用户也可利用该阶段的 shell 命令实现对 UEFI 的配置和检测操作。

● RT 阶段

当系统进入 RT 阶段，就意味着操作系统已经运行起来了。此时，如上文所介绍的，系统中除了运行时服务和运行时驱动以外，其他的程序都已经结束运行，并且其所占用的内存空间都已经被收回。

2.6 本章小结

本章首先对可信计算理论进行了分析研究，重点研究了 TCG 规范中的 TPM 模块，并探讨了其优缺点，指出 TPM 模块在实际应用中的局限性。其次，对本课题的运行环境 UEFI BIOS 进行了阐述，重点研究分析了 UEFI 框架中的三个关键组件。最后，对 UEFI 驱动模式和 UEFI 运行过程进行了讨论。以上研究分析工作有助于深刻理解可信计算理论和 UEFI BIOS 的相关技术细节，并为课题的进一步设计与实现提供理论和技术支撑。

第三章 基于 USB Key 的可信启动架构设计

本章首先阐述了当前系统的安全需求。之后，从设计原则，安全目标和设计思路入手，结合信任链的相关特性，设计了基于 USB Key 的可信启动架构，并阐述了基于 USB Key 可信启动的混合信任链传递流程，以及分析了系统的安全性和可靠性。最后，对其中涉及的关键问题进行了阐述。

3.1 安全需求分析

计算机终端的启动过程是计算机系统建立与运行中的关键一环，如果计算机的启动过程存在安全方面的问题，那么将会给计算机终端系统带来巨大的安全隐患。引导过程的安全是整个系统安全、可信的基础，只有在安全的引导过程下可以为系统建立一个最初的可信的安全计算环境，同时，也才可以为其他的相关安全技术或者策略提供可信保证。

目前，作为新一代的 BIOS——UEFI BIOS，取代传统 BIOS 已经成为了一种趋势。它虽然适应了现代 PC 的发展潮流，但是，它还存在着许多安全问题。其主要原因主要有：

首先，EFI 规范在制定之初并没有考虑安全问题。虽然之后的 UEFI 规范中引入了可信计算思想和 Secure Boot 机制，并且为之提供了一些安全接口，但是，可信计算理论在实际运用中需要可信根 TPM 芯片的支持，然而并不是所有的用户主板内都嵌有该芯片，也就是说这种方法不具有很好的通用性和普适性。另外，Secure Boot 机制需要与操作系统相配合，文献[52]中就阐述了 Secure Boot 机制的一些缺陷和安全漏洞。通过这些缺陷或者安全漏洞，攻击者就可以绕过 Windows 的 PatchGuard 安全保护机制，实现对操作系统内核的劫持。

其次，因为其 UEFI 执行代码并不像传统的 BIOS 那样全都放在主板上的 Flash ROM 中，而是可以存放于其他存储介质上，如硬盘的某个分区或 U 盘等。这样做虽然增强了 UEFI 的可扩展性，但也为系统带来了安全隐患。

再次，目前基于硬件的攻击越来越多，如基于 U 盘的恶意攻击等。而目前针对这类攻击防护主要是基于操作系统。但是，在 UEFI BIOS 的执行过程中，U 盘等设备也可以发挥其作用。因此，只是对系统的固件代码进行完整性防护是不够的，还需要对系统中所连接的硬件进行安全监测。

最后，UEFI 的可执行文件采用高级语言 C 语言编写，虽然大大提高了其易用性，但是同样增加了被恶意篡改和攻击的风险。

综上所述，计算机引导过程的安全可信是整个计算机系统安全可信的基础。而 UEFI 其自身的易用性和可扩展性给系统安全带来了巨大的隐患。虽然 UEFI 提出了一些保护自身安全的策略和技术，但是这些策略和技术在安全性上并没有达到预期的效果。因此，如

何为支持 UEFI 系统的计算机提供一种安全可信的启动保护机制成为了当前研究的一个热点。

基于这种情况,可信计算思想被应用到 UEFI BIOS 的防护过程中,TCG 组织通过可信链技术实现了可信计算思想的部分功能。但是,人们发现可信链中的度量模块 TPM 在实际运用以及推广过程中存在缺陷。因此,需要寻找到一种可信硬件来代替 TPM 行使度量和认证功能,实现一种普适的安全可信的启动保护机制。文献[55]提出利用 USB Key 来代替 TPM 行使 TPM 的职责,并分析了其可行性。

3.2 设计原则与设计思路

基于 USB Key 的可信启动架构设计充分利用了 USB Key 和可信计算的技术优势,同时结合计算机终端启动的一般过程,并考虑到现有安全研究存在的不足。下面首先论述本课题的设计原则、安全目标与设计思路。

3.2.1 设计原则

根据可信计算理论,组件被信任的条件是有如下几点:

- a) 组件身份可确定且合法;
- b) 在没有被非法篡改的情况下,组件是执行过程是可预期的。也就是说,当系统的完整性度量和验证通过时,其行为是一定的、是按照预先设计的流程执行的;
- c) 组件的完整性和可靠性是可以被真实验证的。

可信引导过程需要确保每个组件是被信任的,其实现过程必须建立在一定的安全原则之上,其设计的基本原则主要包括以下几点:

- 1、信任的逐级传递性。信任的传递必须建立在组件验证通过的基础上,因此,在下一个组件执行之前,必须首先对其进行完整性验证,验证通过之后才能交付系统控制权。
- 2、度量和验证方法不能仅仅通过软件实现,必须利用可信硬件实现。
- 3、可信链的建立是以逐级度量和信任传递为基础的,可信硬件必须保证度量过程中所用到的标准值的安全与完整。同时,可信硬件不能提供这些数据的外部访问接口,确保标准值在使用过程中不脱离可信硬件管控范围,以确保标准值不被破坏。

3.2.2 设计目标

基于 USB Key 的可信启动架构的设计目标主要包括以下几点:

- 1、从计算机终端系统启动初期进行保护,建立最初的可信任的运行环境
- 系统的启动过程是计算机最为关键的一环,如果系统的启动过程遭受的攻击或篡改,那么即使在操作系统或者内核层加入各种保护措施或者保护技术,系统也将进处于一个无法预知的环境中。因此,可信启动过程必须能够确保系统以预期的行为启动,保护启动过程的完整性与可靠性,为用户建立一个最初的可信任的运行环境,为其他的安全保护技术

提供有力支撑。

2、针对 UEFI 中 DXE 阶段和 BDS 阶段的脆弱性进行安全防护

在 UEFI 运行的各个阶段中，DXE 阶段是 UEFI 的核心阶段。在这一阶段不仅完成了系统的大部分初始化，同样也为操作系统的引导过程做了必要的准备。DXE 阶段从 DXE Foundation 获得系统控制权之后，就开始加载系统内的所有驱动程序。因此，在这一阶段需要确保驱动程序加载过程的完整性，主要包括以下三点：

● 驱动程序的完整性

为保证驱动程序自身的完整性不受破坏，需确保所加载的驱动程序不被恶意篡改或者感染病毒。

● 过程的完整性

为保证系统加载 UEFI 驱动过程的完整性，须确保加载过程中所需组件不能缺少也不能增多，其加载顺序也不能改变，否则将会使系统进入一个不确定的状态，这与可信计算思想是相违背的。

● 防止未知硬件侵入

因为在 UEFI BIOS 的可扩展性，UEFI BIOS 可以访问所有的分区，包括符合 UEFI 格式的 U 盘等设备。而目前基于硬件的攻击越来越多，这就需要在 UEFI BIOS 阶段确保连接到终端系统上的所有硬件是已知的，防止用户有意或无意的在系统中添加未知硬件对系统造成损害。

3、用户身份合法性以及级别验证

对用户身份进行合法性认证可以在很大程度上避免非法用户对计算机终端的侵入与篡改。同时，对用户身份进行分级管理还可以降低潜在的威胁和不确定性。由此，将合法的用户分为两种：管理员用户和普通用户。管理员用户不仅可以实现对计算机的使用，而且可以对计算机的安全系统进行合法的更新和扩展；而普通用户只能使用计算机。

4、安全进制的引入对系统性能影响小

一方面，在 UEFI 阶段的驱动程序与应用程序需要遵循 UEFI 规范，不对原有系统进行过多的更改，所加模块均以协议的方式实现。另一方面，为了不破坏操作系统原有的保护机制以及便于操作系统之后的更新操作，尽量少的修改操作系统。

基于 USB Key 的可信启动架构的设计将从上述四点出发，并通过这四点检验本课题所研究保护机制的有效性和相对高效性。

3.2.3 设计思路

基于 USB Key 的可信启动架构设计主要集中于以下几个方面：

1、可信硬件的选择

根据可信计算理论，在计算机终端系统中实现可信引导需要可信硬件的支持。目前，大多数的可信引导过程是基于 TPM 实现的。由 2.1.2 节的分析可知，TPM 芯片在实际的运

用中会有一些缺陷，而 USB Key 在这方面具有优越性。

USB Key 是一种带有 USB 接口的具有密码运算功能的高度可靠的智能卡，其自身所固有的安全保护机制，确保了其内部的静态数据和传输过程数据的安全。这主要包括对其内部数据的访问控制以及传输过程的安全保证。对内部数据的访问控制主要体现在对内部数据的管理上，这部分数据主要指的是 PIN 码、加解密的密钥等。这些数据一旦生成就只能在 USB Key 内部应用，不能被读出。对传输过程的安全保证主要体现在 USB Key 与主机传输的数据是经过加密的，使得非法截获者无法得到明文信息，并且能够避免对通信数据的非法篡改。USB Key 的数据传输方式主要有四种：明文传输，密文传输，认证传输和混合传输。用户可以根据实际需要选择合适的传输方式。

此外，USB Key 还具有便于携带；难以破译和伪造，可靠性高；快速计算，存储量大等优点，可以实现身份认证，完整性度量以及数据加解密等功能。其硬件结构如图 11 所示，由图可知，USB Key 主要包含了 I/O，MPU，CAU，RNG，SAL，SRAM，FLASH，USB CONTROL 和 JTAG INTERFACE 等部分，其各自功能如下：

I/O（输入输出接口）提供了 USB Key 与主机的通信通道。

MPU（微控制器）是 USB Key 的核心部分，实现对整个 USB Key 的控制，完成基本的指令以及存储和逻辑控制等。

CAU（密码协处理器）主要负责协助 MPU 实现对 RSA 的相应处理操作。

RNG（随机数发生器）主要用来产生一组定长的随机数。

SAL（安全访问逻辑）为 USB Key 自定义的硬件安全逻辑。

SRAM（易失性存储器）和 FLASH（非易失性存储器）为 USB Key 提供了存储区域，使 USB Key 可以完成对数据的存储操作。

USB CONTROL（USB 控制器）是一种通过硬件控制 USB Key 和主机间的传输通信的逻辑单元。

JTAG INTERFACE（调试接口 JTAG）为开发人员提供了接口，该接口对用户不可见。

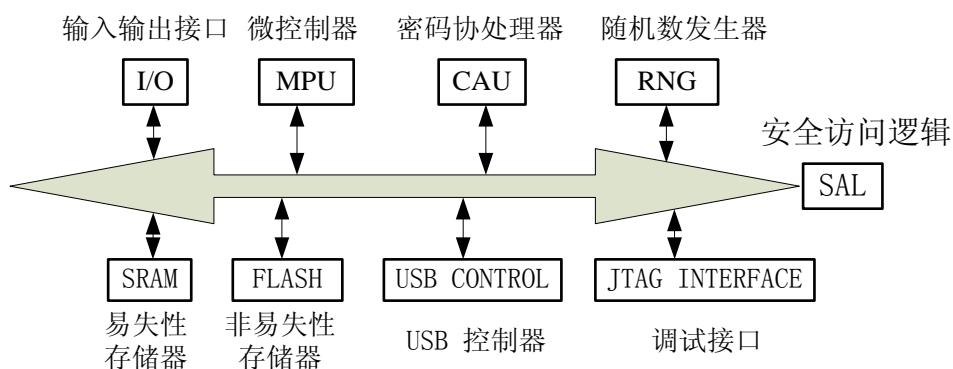


图 11 USB Key 硬件结构示意图

对比 USB Key 和 TPM 的硬件结构示意图（图 3）可以发现，两者在硬件结构上有很

多相似的模块。并且，与 TPM 相比，USB Key 是一种可插拔的设备，与主机采用松散耦合的方式连接，其应用范围更大更广。并且，USB Key 在身份认证方面具有较大的优势，可以对用户进行分级管理，符合安全设计目标。因此，本方案采用 USB Key 作为可信根，并通过 USB Key 的相关功能实现终端系统的可信启动过程。

2、可信基

所有安全系统都会面临一个问题，即如何保证自身的安全模块的安全。因此，可信计算提出了可信基 (Trust Computing Base, TCB) 概念。实现系统的可信启动过程需要 TCB，只要 TCB 不被攻破，那么就能认为该启动过程是安全的和可信的。TCG 制定的可信计算标准阐述了如何利用 TPM 来保护终端平台的安全。

由于 USB Key 是被动传输设备，需要主机端提供必要的支持，既 USB Key 只有在主机端驱动运行的情况下才能运行，所以需要把 UEFI 运行的前两个阶段 (SEC 阶段和 PEI 阶段) 也归结到可信基的范围中去。可以这么做的原因是：UEFI 的扩展功能主要体现在 DXE 阶段和 BDS 阶段，其主要的安全威胁也来源于这两个阶段 (如 Bootkit 等)。因此，相比 TPM 作为可信根，利用前两个阶段和可信硬件 USB Key 一起作为可信根也不会明显降低系统的有效性。

3、度量点的选取

本课题基于 Windows 7 操作系统研发，目前应用较为广泛的 Windows 7 系统的，其一般启动过程为：UEFI BIOS->MBR->Bootmgr (启动管理器) -> Winload.exe->内核加载。

根据可信启动过程，系统的效率主要受度量粒度和 USB Key 计算能力与传输效率的制约。而度量粒度关乎到系统安全的程度，它跟度量点的选取有关；USB Key 的计算能力与传输效率受限于自身的硬件条件。所以，度量点的选取需要考虑安全和效率的平衡。为此，课题需要根据启动过程中不同阶段的自身特点，采取不同的度量点。

4、用户身份认证

在系统进入 BDS 阶段后，用户可以选择进入 UEFI shell 平台，并在 shell 平台上选择驱动程序或者应用程序的加载与删除，也可以实现对 UEFI 的重新配置。用户也可以在这一阶段选择所要启动的操作系统和引导路径。因此，在系统进入 BDS 阶段后，首先应该对用户身份进行合法性验证。同时，对于单位用户来说，更希望对单位内部用户进行分级管理，普通用户只能使用自己的计算机，而不能对计算机的安全系统进行更改；而管理员用户则可以在使用计算机的同时实现对安全系统的更新和扩展。这就需要安全系统能够在用户合法性验证通过的基础上实现对用户身份的级别进行鉴别。

传统的身份认证过程一般采用口令方式进行。这种认证方式安全性较低，非法用户可以通过穷举的方式来破解口令。一旦口令被破解，非法用户就可以登录进系统中实施各种非法操作。若系统采用 USB Key 作为可信硬件，那么就可利用基于 USB Key 的身份认证技术，从而加大非法用户登录系统的难度。

每一个 USB Key 都具有硬件 PIN 码保护，PIN 码和硬件构成了用户使用 USB Key 的两个必要因素，即“双因子认证”。用户只有同时取得了 USB Key 和用户 PIN 码，才可以登录系统，如此可大大降低非法用户破解 PIN 码的可能性。

5、高效可靠的信任链结构

由可信计算理论可以知道，信任在传递过程中会出现一定的信任损耗，而传统的信任链过程因为信任链较长可能会出现信任损耗较大的情况。并且，信任链过长也不利于系统的维护。由此，人们提出了星形结构的信任链，与传统的链式过程相比，星形结构的信任链信任损耗低、效率更高。但是，使用该信任链结构需要对系统的启动过程做出修改。而可信硬件为被动设备 USB Key，需要 UEFI BIOS 提供支持，并且 UEFI BIOS 各个运行阶段联系相对紧密，因此，在 UEFI 运行过程中不采用利用星形结构。为此，可信启动的信任链过程需要依据终端启动的不同阶段采用不同的链式结构。

3.3 基于 USB Key 的可信启动系统框架

根据上述的安全目标和设计思路，可以构建出一个基于 USB Key 的可信启动架构，以保障计算机终端系统启动过程的安全可信，为用户建立一个最初的可信计算环境。

3.3.1 基于 USB Key 的可信启动架构

因为本系统架构以 USB Key 为可信硬件，考虑到 USB Key 设备为被动设备，且 UEFI 系统中并没有专有的 USB Key 驱动，因此为了最大限度的保证系统启动过程的可信，需要在 UEFI BIOS 中添加 UEFI 驱动程序实现，以实现对 USB Key 设备的识别与访问，并设计基于 USB Key 的具有度量功能和认证功能的应用程序，以实现对 USB Key 具体命令的操作，完成系统的完整性度量与验证过程 and 用户合法性认证过程。由此本课题设计的基于 USB Key 的可信启动架构如图 12 所示，其主要由可信基和可信机制层两部分构成。其中，可信机制层主要包括驱动模块、认证模块、度量模块、恢复模块和基准库五部分；可信基则是由 UEFI 的前两个阶段和可信硬件 USB Key 组成。

其中，可信机制层中的认证模块、度量模块和 USB Key 驱动模块为系统的关键模块，而恢复模块和基准库是系统的辅助模块，这两个辅助模块的存在是为了使整个系统更加完整健壮。其各个模块的主要功能是：USB Key 驱动模块负责实现 UEFI 环境下 USB Key 的驱动程序，使 USB Key 设备在 UEFI 环境中可识别，可使用；认证模块负责实现可信启动过程中对用户身份的合法性认证以及对用户身份的级别进行鉴别；度量模块主要是利用 hash 算法对启动过程中的各个组件进行完整性度量和验证；在可信启动过程中出现验证错误或者其他未知错误时，则调用恢复模块对系统进行恢复，以保证系统运行的安全可靠。基准库则存放着各个组件和硬件白名单及其各自的 hash 标准值，可以为度量模块的验证提供可靠的依据。

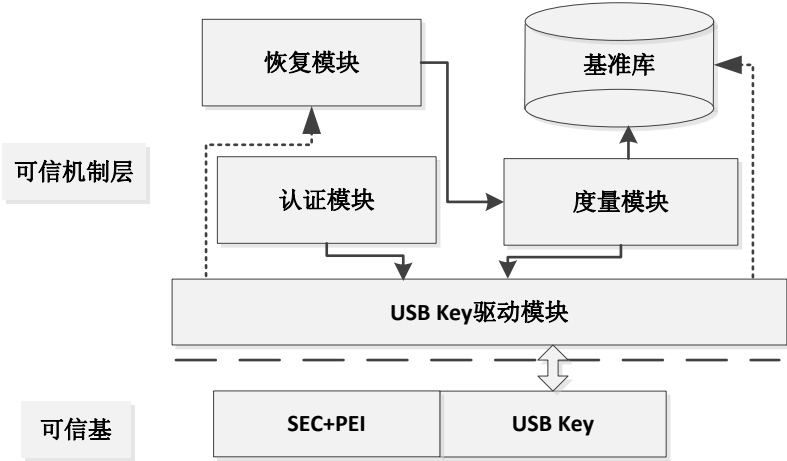


图 12 基于 USB Key 的可信启动架构

3.3.2 基于 USB Key 的可信启动过程

结合本文所设计的基于 USB Key 的可信启动架构和终端系统启动的一般流程可以得到基于 USB Key 的可信启动的工作过程，如图 13 所示。

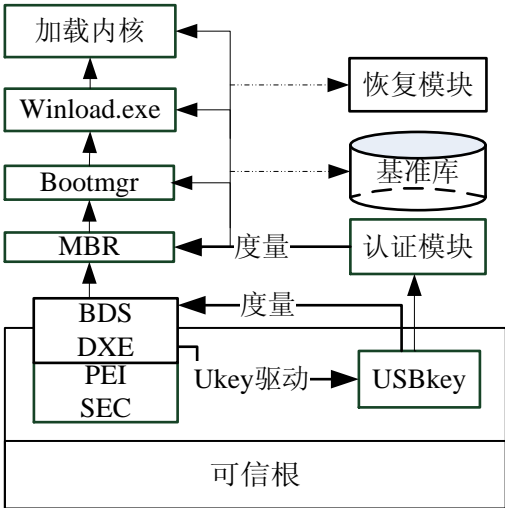


图 13 基于 USB Key 可信启动过程

因为 UEFI BIOS 和操作系统各自的特殊性，本系统的可信启动过程采用混合信任链方式。其中，在 UEFI BIOS 启动过程，系统采用链式的信任链过程过程；在操作系统引导阶段，系统采用星形的信任链过程。当机器加电后，主机进行硬件自检，之后开始进行系统的启动过程。当 UEFI BIOS 运行到 DXE 阶段后，首先利用 DXE 阶段的 dispatcher（分发器）加载 USB Key 驱动程序，在 USB Key 驱动程序成功加载后，开始系统的可信引导过程。下面分别对 UEFI 启动阶段和操作系统阶段的可信启动过程进行分析。

1、UEFI 运行阶段

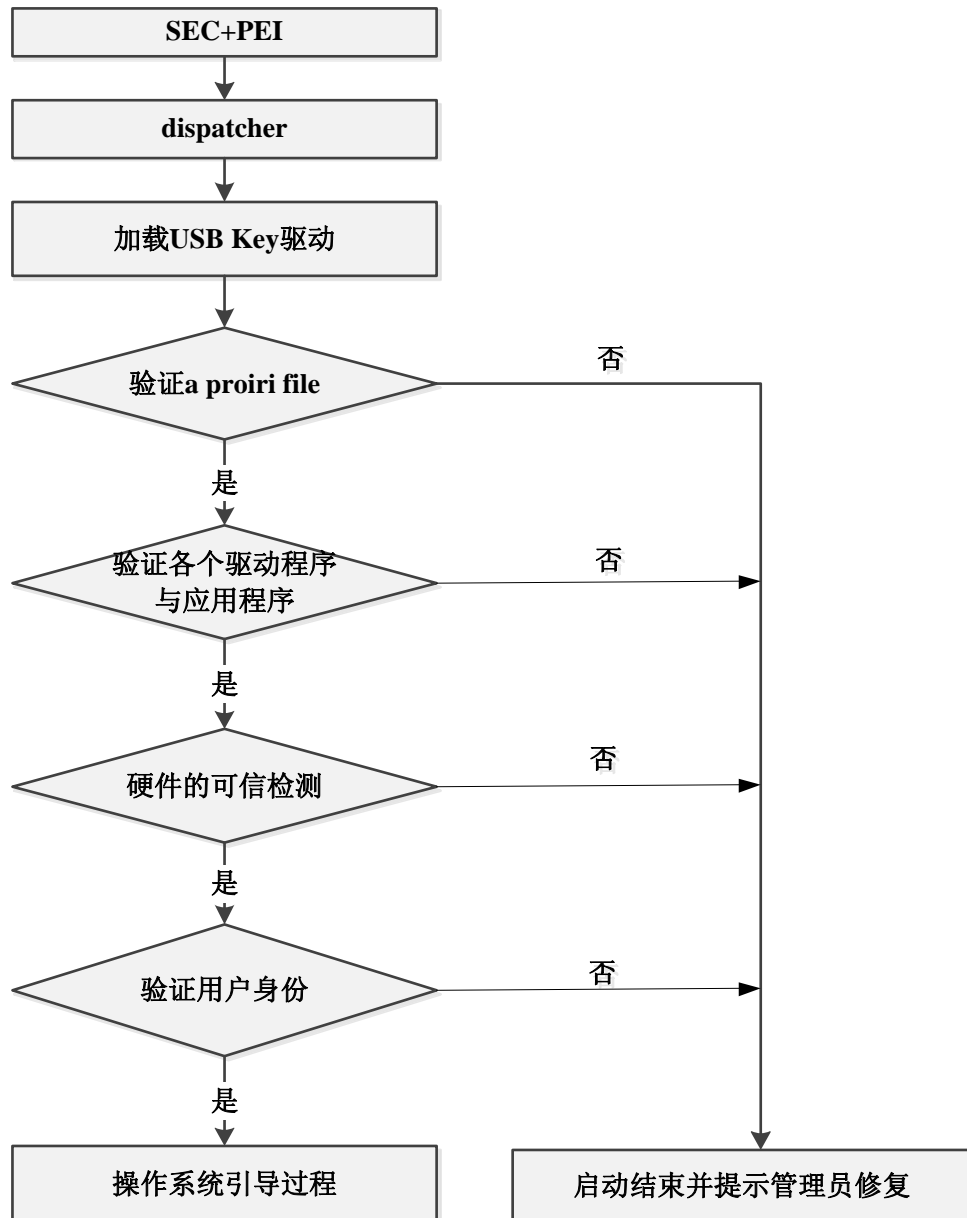


图 14 UEFI 阶段链式信任链流程图

UEFI 运行阶段采用链式的信任链过程，如图 14 所示。在 DXE 阶段，需要保证驱动程序的完整性以及驱动程序是按照预期的顺序进行加载的。驱动程序的加载顺序是由 BFV 中的启动顺序管理文件、驱动程序中的依赖表达式决定的。DXE 分发器首先找到启动顺序管理文件中列出的驱动程序，使其按顺序加载。之后，再寻找剩余的驱动程序，检测其依赖表达式，当其依赖表达式满足时，才能加载到内存中。因此，首先需要对启动顺序管理文件进行完整性度量并与基准库中的标准值进行比较，如果验证通过则说明驱动程序顺序没有被更改。之后，对每一个要加载的驱动程序在加载之前进行完整性验证，来确保所要加载的驱动程序是完整的、可信的。

当所有的驱动程序加载完成后，系统的控制权将转交给 BDS 阶段，在这一阶段，首

先，需要对用户 USB Key 的合法性进行鉴别。如果该用户 USB Key 是合法的，则开始鉴别该用户 USB Key 的级别，并确定该用户的权限，之后，对相应级别的用户身份进行认证。认证通过后，开始对这一阶段运行过程中所用到的应用程序以及 UEFI shell 进行完整性度量，只有在完整性验证通过的情况下才能够将该应用程序加载到内存中执行。

2、操作系统引导阶段：

在 UEFI 完成完整性度量之后，便开始操作系统的引导过程。从这一部分开始，信任链选择使用星形信任链过程，如图 15 所示，这样做的好处主要有两点：1) 减少信任损耗；2) 可以尽可能减少对操作系统的修改，这样可以确保操作系统内部原有的保护机制可以有效运行，也有利于在操作系统的更新或添加新功能时对系统进行扩展。

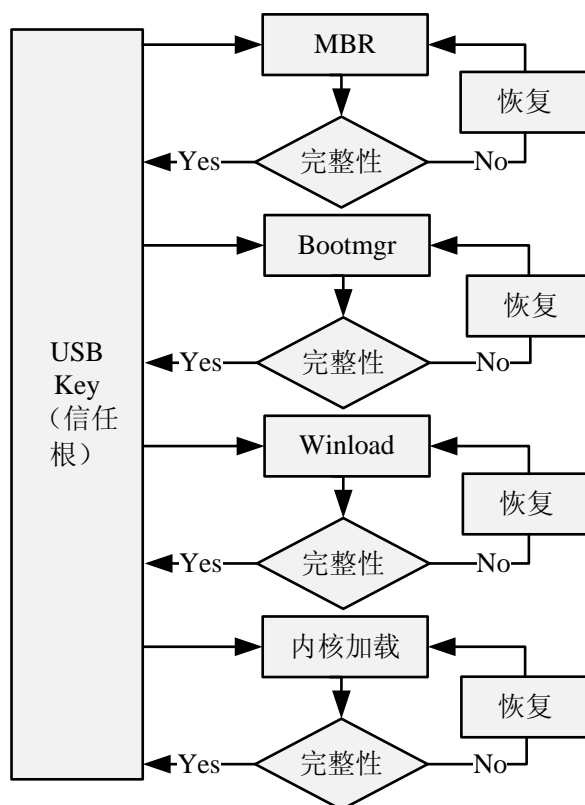


图 15 操作系统阶段星形信任链流程图

由图 15 可知，系统在将系统的控制权转交给 MBR 之前，首先利用可信硬件 USB Key 分别对 MBR，Bootmgr，Winload.exe 和内核加载程序进行完整性度量和验证，如果完整性验证不通过，则表明该模块处于不安全的状态，那么系统需要通过恢复模块，对该模块进行恢复。只有在所有模块都验证通过的情况下，才能开始操作系统的引导与加载过程。这样既保证了操作系统的完整性和原有的安全进制不被破坏，同时可以提高系统的运行效率，达到系统安全与系统运行效率的统一。

3.3.3 系统安全性与可靠性分析

(1) 信任根的安全性。USB Key 代替 TPM 作为信任链的信任根，它主要从两个方面

保障了自身的安全性：一是 USB Key 自身具有计算功能，能够在 USB Key 内部进行独立的运算，并且内部的私有数据都不被导出；二是，USB Key 与主机的通信过程都不是明文的传输方式，确保了数据在传输过程中不被窃取。

(2) 混合信任链过程的安全性。本课题基于可信计算理论，设计了基于 USB Key 的混合信任链启动过程，混合信任链过程相对于传统的链式信任链过程，其信任损耗小，这就增加了系统的可信性和可靠性。另外，在混合信任链过程中，每个组件执行之前都要通过可信硬件进行完整性度量和验证，确保所要执行的组件是完整的、没有被篡改的，确保启动过程是按照系统预期运行的。

(3) 认证过程的安全性。利用 USB Key 的双因子身份认证功能，可以大大增强认证的可信性和抗攻击性。首先，通过 USB Key 和主机的挑战应答，保证了插入主机中的 USB Key 是确定的、合法的。其次，通过 PIN 码对用户身份进行认证，PIN 码的输入错误次数不能超过 5 次，有效地降低了非法用户通过暴力破解登陆系统的可能性。

(4) 基准库与恢复模块的安全性。系统通过 USB Key 对基准库进行签名保护，其私钥存储在 USB Key 不能被取出，由此，保证了基准库不被破坏。而恢复模块则存储在管理员 USB Key 的 usb 存储盘中，与主机物理隔离，确保其安全性。

3.4 关键问题的提出

从前述分析可知，实现基于 USB Key 的可信启动系统所要的关键问题主要有：UEFI 下 USB Key 的驱动、基于 USB Key 的完整性度量与认证、标准值的存储与封装、度量点的选取等。

3.4.1 UEFI 下 USB Key 的驱动

由于 USB Key 设备最初的设计主要是应用于操作系统（Windows 或 Linux）层面上，所以在 UEFI BIOS 中并没有 USB Key 的专有驱动程序。由上文中所论述的设计目标和原则可知，为了增强 UEFI BIOS 的安全性，最大限度保障启动过程的可信性，需要在 UEFI 启动过程中就实现对可信硬件 USB Key 的使用，以完成对 UEFI BIOS 的完整性度量和验证以及用户合法性认证操作。因此，需要实现 UEFI 环境下的 USB Key 驱动，以实现在 UEFI 环境中对 USB Key 设备的识别与访问。

UEFI 通过对 USB 系统软件层的实现提供了对 USB 类设备的支持，并且为便于开发人员的对 UEFI 的扩展，UEFI 规范提供了统一的 UEFI 驱动模式。依据 UEFI 驱动模式，可以实现 USB Key 在 UEFI 下的驱动程序，并通过中断传输方式实现主机与 USB Key 之间的通信过程。

3.4.2 基于 USB Key 的完整性度量与认证

在 UEFI 系统中，其驱动程序与应用程序是以可执行文件的形式存储的，因此系统的

完整性度量操作归根到底是实现对可执行文件的完整性度量和验证，本文主要利用了 USB Key 的 sha-1 运算实现对可执行文件的完整性度量与验证。具体过程如下：

首先，在一个纯净系统中利用 USB Key 的 sha-1 运算实现对基准值的采集，并通过 USB Key 的 RSA 加密算法实现对基准值的签名运算，其私钥存储在 USB Key 中，且不能被读出。在得到系统的基准值之后，就可以利用 USB Key 实现对系统启动过程的完整性度量和验证操作。在 USB Key 取得系统控制权后，首先对要度量的模块进行 hash 运算，得到散列值，之后利用存储在 USB Key 内私钥对基准值进行解密，得到对应模块的基准值。最后比较散列值和基准值，验证该模块的完整性。如果完整性没有被破坏，则继续执行；否则，则停止启动过程。

本课题采用的基于 USB Key 和 PIN 码的双因子身份认证可以大大降低非法用户入侵的可能性。由于身份认证过程涉及到 PIN 码的输入，需要其他设备如键盘等的支持，因此，可以在 BDS 阶段建立身份认证模块，利用挑战应答（challenge-response）模式实现用户身份认证过程，具体如下：

在进入 BDS 阶段之后，身份认证模块获得系统的控制权，首先，在主机端生成一个随机数，并由 USB Key 利用约定好的密钥对其加密，在主机端对该密文解密得到一个数值并验证是否与生成的随机数一致。一致则主机对 USB Key 挑战成功，否则失败。同样，也需要利用 USB Key 产生一个随机数，向主机发起挑战，与上一过程类似。结果一致则表明验证成功，否则返回失败并关机。

随后，提示用户输入 PIN 码，PIN 码验证成功后则进入下一步，否则提示错误，并要求用户再次输入。如果输入 5 次仍然验证不通过，则 USB Key 被锁死，并且只能由管理员对其进行解锁。

3.4.3 标准值的存储与封装

基准值为完整性度量和验证功能提供了依据。它关系着整个可信系统的可靠性和安全性。另外，基准值在系统内以何种结构进行组织和存储，才能降低基准值搜索和动态匹配的时间复杂度，并兼顾空间效率，对于完整性度量系统是非常重要的。因此，如何在确保标准值不被破坏的前提下，提高系统运行效率，需要研究标准值的存储与封装过程相关技术。

3.4.4 度量点的选取

所谓度量点，即为监控对象中会触发完整性度量的节点（某条指令或其他特殊位置）。如果度量点设置太频繁，如采用每执行一条指令度量一次的方式，那么系统的时间开销将令人无法接受；如果设置太稀疏，则系统的可信性会大大降低。所以度量点如何选取、度量密度如何设置是影响系统实现效果的一个关键问题。

因为本课题是基于目前应用最为广泛的 Windows 7 系统，根据实际运行情况的不同，

拟采取不同的度量点：

对于启动顺序管理文件。该文件用于保证 DXE 阶段驱动加载的顺序，根据它在 UEFI 中的特殊性，应该在 USB Key 获取系统控制权之后，首先对其进行完整性验证，以确保驱动程序是按照预期的顺序执行的。

对于驱动程序。因为驱动程序需要通过 LoadImage（）加载到内存中，所以可以在 LoadImage（）中加入拦截代码，使驱动程序在加载到内存之前先进行完整性验证，在完整性验证通过之后才能加载到内存中执行。

对于硬件的可信检测。硬件的初始化需要借助于其相应的 UEFI 驱动程序，当一个新的硬件连接到计算机上时，UEFI BIOS 在启动过程中会为其分配一个设备 Handle，并检测是否有驱动支持给设备，如果有，则执行该驱动程序实现硬件的初始化。因此，可以考虑利用驱动程序实现对硬件的可信检测。首先，收集合法硬件的相关信息，并对其进行 hash 运算，用该抽样值建立一个硬件白名单。其次，当一个系统检测到某个驱动支持的硬件设备存在时，利用驱动中的接口描述符等相关函数收集硬件相关信息，并进行 hash 运算。查找比较硬件白名单，检测其否在白名单中，如果在，则硬件合法；如果不在，则提示出现未知硬件，提醒管理员。

对于 MBR，bootmgr，Winload.exe 和内核加载过程。在考虑到 Hash 计算的特性和不破坏原有操作系统的原则，对操作系统引导过程各个阶段的完整性验证过程在 BDS 阶段执行。这样做是因为 Windows 7 操作系统自身具有一定的安全机制，且不断更新，如果把度量代码加入到操作系统引导过程中，就需要不断地根据操作系统的更新情况而做出相应的改变，这无疑将增大度量的难度和工作量。因此在该度量过程中，采取星形的可信链方式，在所有模块都通过验证的情况下才能开始操作系统的引导过程。

3.5 本章小结

为了设计与实现基于 USB Key 的可信启动系统，本章首先分析了当前安全的需求，并结合系统的设计原则、设计目标以及设计思路，设计了基于 USB Key 的可信启动系统架构。随后，依据 2.1.1 中的可信计算理论，为了减少信任在传递过程中的损耗，兼顾安全与效率的统一，设计了基于 USB Key 的可信启动的混合信任链结构，并对其过程进行了阐述。最后，对系统实现过程中所涉及的关键问题进行了分析。

第四章 基于 UEFI 驱动模式的 USB Key 驱动

为了实现基于 USB Key 的可信启动系统，首先需要确保 USB Key 在 UEFI BIOS 环境中可以被识别、访问。为此，需要设计和实现 UEFI 环境下的 USB Key 驱动程序。本章首先概括地介绍了 USB 的系统结构，随后对 UEFI 系统中已存在的 USB 驱动栈进行简要分析，最后根据 UEFI 驱动模式设计 UEFI 环境下的 USB Key 驱动程序，并对该程序和相关协议进行了编码实现。

4.1 USB 系统结构

USB 的系统结构是由主机，外围设备以及它们之间通信所依赖的软件和协议共同构成的。USB 系统框图如图 16 所示。

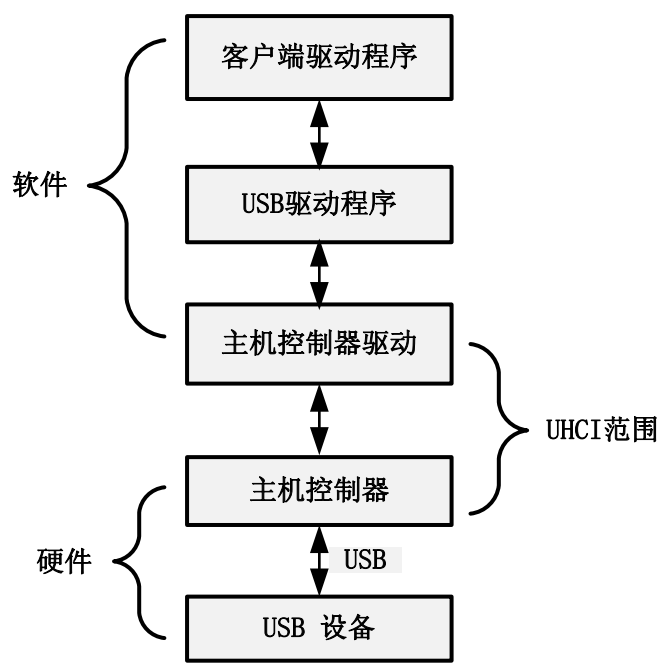


图 16 USB 系统框图

由图 16 可知，在 USB 系统结构中包含了一系列的硬件和软件。其中，硬件主要包括：主机控制器（Universal Host Controller，HC）和 USB 设备（USB Device）；软件主要有：客户端驱动程序（Client Driver Software）、主机控制器驱动（Host Controller Driver，HCD）和 USB 驱动程序（Universal Serial Bus Driver，USB D）。具体如下：

- 主机控制器

目前，计算机系统中主要存在两种主机控制器接口规范：UHCI（Universal Host Controller Interface，通用主机控制器接口）和 OHCI（Open Host Controller Interface，开放主机控制器接口），分别由 Intel 和 Microsoft 提供主要的技术支持。

- 主机控制器驱动程序

可以用来实现对主机控制器的管理与配置。

- 客户端驱动程序

客户端驱动程序由操作系统或者是 USB 厂商提供,其主要功能是实现对特定设备功能的管理以及配置等操作。它不能实现对设备硬件的直接访问,而是需要借助于主机控制器和的协议软件实现与设备中的某个接口进行通信,这一点明显区别于其他的非 USB 驱动程序。

- USB 驱动程序

在特定操作系统中为与之相连的客户端驱动程序提供了面向功能的接口设计,通过这种接口设计简化了驱动程序的设计。

由 USB 系统结构图可以知道,用户的应用程序需要借助 USB 驱动程序访问 USB 设备,而 USB 驱动程序是建立在通用的 USB 主机控制器接口之上的。

4.2 UEFI USB 驱动程序分析

通过对 USB 系统软件层的实现,UEFI 系统提供了对 USB 类设备的支持。图 17 显示了 UEFI 提供的 USB 驱动栈。由图 17 可知,其 USB 驱动栈主要包括:USB 主机控制器驱动程序,USB 总线驱动程序以及一些相关的设备驱动程序。因为本课题在驱动程序的开发过程中只需要调用驱动栈各层驱动程序的相关接口,而不需要了解其具体的实现过程,因此,只对驱动栈各层的功能做简要介绍。

1、USB 主机控制器驱动程序

USB 主机控制器驱动程序依赖于 USB 主机控制器的规范,在 UEFI 系统中,USB 主机控制器驱动程序是针对 UHCI 设计的。它通过在 USB 上处理相应的数据结构,使得数据能够在系统内存与 USB 设备之间传递。该驱动程序主要功能有:

- 初始化 USB 主机控制器,并实现相关配置工作,如分配内存,建立 UHCI 驱动程序的私有数据和设置中断传输间隔时间等等;
- 通过利用 EFI_PCI_IO_PROTOCOL 协议生成 EFI_USB_HC_PROTOCOL 协议实例;
- 通过 EFI_USB_HC_PROTOCOL 协议,为 USB 主机控制器提供 I/O 抽象,并为 USB 总线提供抽象接口。

2、USB 总线驱动程序

USB 总线驱动程序使用 EFI_USB_HC_PROTOCOL 协议,为 USB 总线上的每一个 USB 设备产生一个独特的子句柄,并在这些子句柄上安装 EFI_DEVICE_PATH_PROTOCOL 协议和 EFI_USB_IO_PROTOCOL 协议。这样就可以实现 USB 设备与 USB 主机控制器之间的通信。

EFI_USB_IO_PROTOCOL 协议可以实现对 USB 设备的管理以及通信。该协议提供了

访问设备的传输服务和管理设备服务。通过对该协议的使用可以实现对 USB 设备传输类型的设置，并通过该协议可以进行获取和设置设备属性的相关操作。设计和实现 USB Key 驱动程序时，会用到该协议的相关功能。

3、设备驱动程序

USB 设备驱动程序通过利用 EFI_USB_IO_PROTOCOL 协议为具体的 USB 设备产生 I/O 抽象和某些服务协议，如 USB Mouse 驱动程序为 USB 鼠标提供 EFI_SIMPLE_POINTER_PROTOCOL 协议。在 UEFI 系统中提供了 USB 大容量存储设备驱动（USB Mass Storage Device Driver），USB 键盘驱动（USB Keyboard Device Driver）和 USB 鼠标驱动（USB Mouse Device Driver）三种设备驱动程序，可以为 USB Key 驱动程序的设计与开发提供一定的借鉴，便于 USB Key 驱动程序的设计与开发。

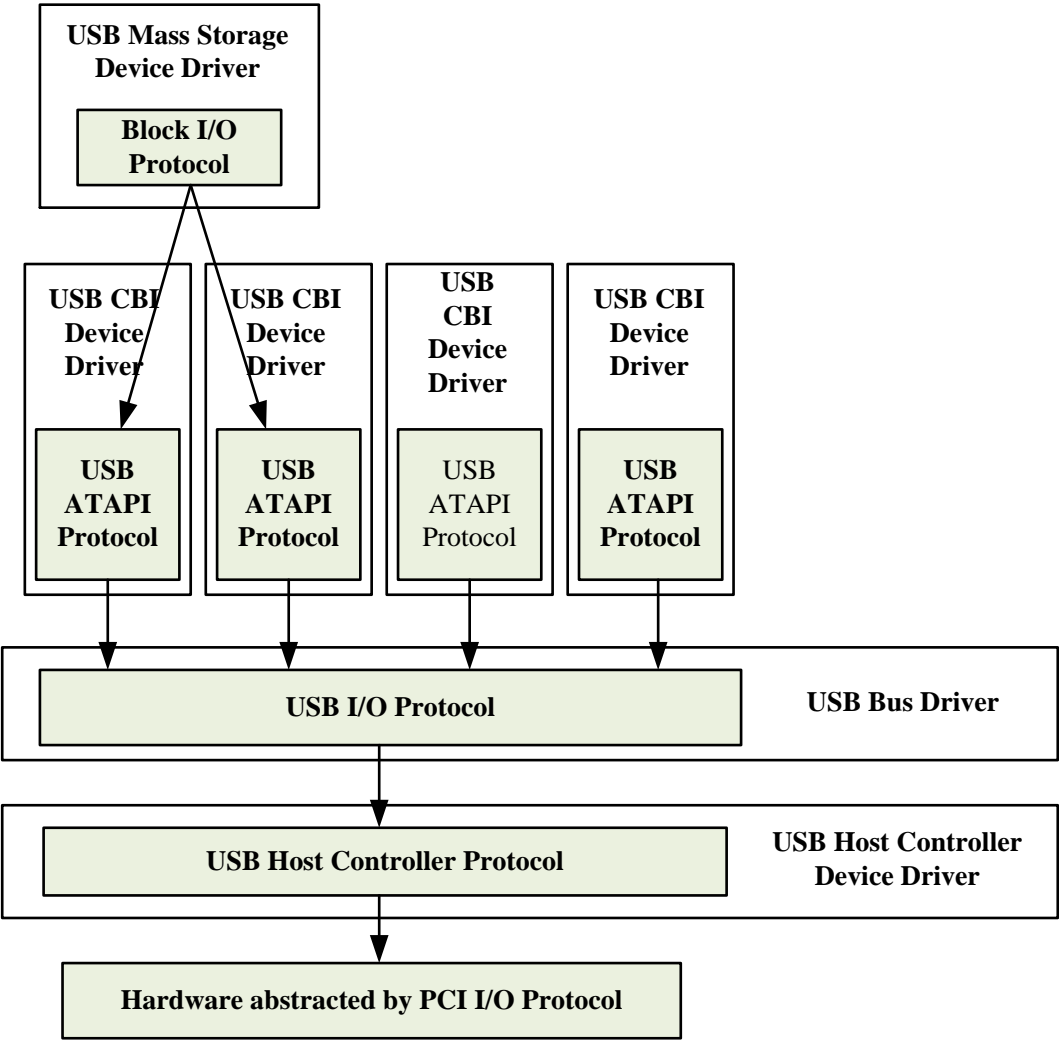


图 17 UEFI USB 驱动栈

4.3 基于 UEFI 驱动模式的 USB Key 驱动程序的设计与实现

虽然在新一代的 UEFI 系统中提供了芯片智能卡接口设备驱动 CCID（USB Chip/Smart

Card Interface Devices-USB), 可用于实现对 USB Key 的识别与管理, 但是 UEFI 组织并没有提供该驱动程序的源码, 也没有提供相应的接口。因此, 需要自行实现 USB Key 的驱动。

通过对 UEFI 环境中的 USB 驱动栈分析可以知道, USB Key 程序位于 USB 驱动栈的最顶层。需要 USB Host Controller 驱动程序和 USB BUS 驱动程序的相关支持。下面将论述 UEFI 环境下的 USB Key 的具体实现过程。

4.3.1 驱动程序的关键函数

依据 UEFI 驱动模式, 实现 UEFI 环境下的 USB Key 的驱动程序, 必须实现驱动程序的入口函数 `EFI_DRIVER_ENTRYPOINT` 和符合 UEFI 驱动模式的协议 `EFI_DRIVER_BINDING_PROTOCOL`, 另外, 为了更好地管理 USB Key 设备, 可以实现 `EFI_COMPONENT_NAME_PROTOCOL`, 利用该协议可以在 UEFI 环境中以自定义的标示符表示该设备, 如表 2 所示。同时, 为了避免驱动程序以及使用该驱动程序的相关程序反复对 USB Key I/O 命令的反复调用, 将 USB Key 相关命令进行抽象, 并以协议的方式实现, 这样有助于提高系统的运行效率和便于之后的扩展。

表 2 USB Key 驱动程序中需要实现协议与函数

协议/函数名	功能描述
USBKeyEntryPoint	USBKEY 驱动程序的入口点。该程序主要功能是在该驱动程序的映像句柄上安装 <code>EFI_DRIVER_BINDING_PROTOCOL</code> 与 <code>EFI_COMPONENT_NAME_PROTOCOL</code> 。
USBKEY_DRIVER_BINDING_PROTOCOL	基于 UEFI 驱动模式的驱动程序所必须实现的协议, 主要用来测试驱动程序是否支持该设备, 以及实现开始和停止驱动程序对该设备的管理
EFI_COMPONENT_NAME_PROTOCOL	该协议为可选的协议, 主要功能是实现 USBKEY 设备在 UEFI 中能够以自定义的标识符表示该设备。
USBKEY_CMD_PROTOCOL	自定义协议, 在 <code>EFI_DRIVER_BINDING_PROTOCOL</code> 中实现, 实现对 USBKEY 设备的 I/O 抽象

4.3.2 `EFI_DRIVER_BINDING_PROTOCOL` 的实现

`USBKEY_DRIVER_BINDING_PROTOCOL` 是 UEFI 驱动模式下必须实现的协议, 主

要工作是测试驱动程序是否支持 USB Key 设备，并对 USB Key 设备进行相应的管理。具体需要实现 USBKeyDriverBindingSupported、USBKeyDriverBindingStart 和 USBKeyDriverBindingStop 三个函数。

1、USBKeyDriverBindingSupported

此函数通过检测传入的句柄，来确定该设备是否为 USB Key 设备。其函数的定义如图 18 所示。

```
EFI_STATUS
EFIAPI
USBKeyDriverBindingSupported (
    IN EFI_DRIVER_BINDING_PROTOCOL *This,
    IN EFI_HANDLE                  Controller,
    IN EFI_DEVICE_PATH_PROTOCOL    *RemainingDevicePath
);
```

图 18 USBKeyDriverBindingSupported 定义

其中，*This 是指向 EFI_DRIVER_BINDING_PROTOCOL 实例的指针；Controller 是传入的设备句柄；而*RemainDevicePath 在本驱动程序中忽略。函数的返回值 EFI_STATUS 为 Success 或 False。

其函数功能的实现过程如下：

1) USB Key 设备是 USB 设备的一种，因此，首先检测该设备句柄是否为 USB 设备句柄。因为凡是 USB 设备，USB 总线驱动程序都会为其创建一个句柄，并安装 EFI_USB_IO_PROTOCOL 协议。因此，利用 UEFI 引导服务的 OpenProtocol 尝试在传入的设备句柄 Controller 上打开 EFI_USB_IO_PROTOCOL 协议，打开成功则说明该设备为 USB 设备。否则，则不是 USB 设备，更不是 USB Key 设备，函数返回错误。

2) 如果是 USB 设备，则利用 UsbGetInterfaceDescriptor()服务获取 USB 设备的接口描述符，包括 InterfaceClass，InterfaceSubClass 和 InterfaceProtocol。并依据这三个参数判断该 USB 设备是否为 USB Key。如果不是 USB Key 设备，函数返回 False。如果是 USB Key，则进一步利用 DeviceDescriptor()检测其 IDVendor、IDProduct 和 BCDDDevice 等参数，从而确定是否是指定的 USB Key，并区分是管理员属性还是用户属性。其部分伪代码如下：

```

Status = UsbIo->UsbGetInterfaceDescriptor (UsbIo, &Interface);
If (InterFaceDescriptor. InterFace class==USB_Key_Class)
    USB Key 设备;
else    return  FALSE;
Status = UsbIo->UsbGetDeviceDescriptor (UsbIo, &DevDesc);
if (UsbClass->VendorId != 管理员.IdVendor) {
    if (UsbClass->ProductId != 管理员.IdProduct) {
        该 USB Key 为管理员 USB Key; }
    else return FALSE; }
if(UsbClass->VendorId != 用户.IdVendor) {
    if (UsbClass->ProductId != 用户.IdProduct) {
        该 USB Key 为用户 USB Key; }
    else return FALSE;
else{ return FALSE; }
}

```

3) 关闭上述过程中打开的协议。并且, 如果函数返回成功, 则开始 USB Key 设备的初始化等相关管理工作。

2、USBKeyDriverBindingStart

该函数主要负责实现对 USB Key 设备的初始化以及相关的管理控制操作。其函数的定义如图 19 所示。

```

EFI_STATUS
EFIAPI
USBKeyDriverBindingStart (
    IN EFI_DRIVER_BINDING_PROTOCOL *This,
    IN EFI_HANDLE                  Controller,
    IN EFI_DEVICE_PATH_PROTOCOL    *RemainingDevicePath
);

```

图 19 USBKeyDriverBindingStart 定义

其中, *This 是指向 EFI_DRIVER_BINDING_PROTOCOL 实例的指针; Controller 是传入的 USB Key 设备句柄; 而*RemainDevicePath 在本驱动程序中忽略。函数的返回值 EFI_STATUS 为 Success 或 False。

其函数功能的实现过程如下:

1) 以 BY_DRIVER 的方式在设备句柄 Controller 上打开 EFI_USB_IO_PROTOCOL 协

议。以实现在本函数中可以调用该协议提供的服务。

2) 调用 `UsbGetInterfaceDescriptor()` 和 `UsbGetEndpointDescriptor()` 函数，分别得到 `interface` 和 `endpoint` 描述符，并通过遍历 `endpoint` 过程找到用于传输中断的 `endpoint`，实现 USB Key 设备与主机的抽象 I/O 操作，利用异步中断实现对设备的管理。其部分伪代码过程：

```

if ((EndpointDescriptor.Attributes 为中断传输端口) {
    if (Endpoint 为输入端口){
        UsbKeyDevice->InEndpointDescriptor=(EFI_USB_ENDPOINT_DESCRIPTOR
        *) (UsbKeyDevice+1);
        为 USBKeyDevice 分配内存并设置命令参数;
    }
    if (EndPoint 为输出端口) {
        UsbKeyDevice->OutEndpointDescriptor = (EFI_USB_ENDPOINT_DESCRIPTOR*)
        (UsbKeyDevice + 1) + 1;
        为 USBKeyDevice 分配内存并设置命令参数;
    }
}

```

3) 实现 `USBKEY_CMD_PROTOCOL`，实现对 I/O 的抽象。其函数定义如下：

```

struct _USBKEY_CMD_PROTOCOL {
    USBKey_INIT_TRANSPORT    Init;
    USBKEY_CMD_PROTOCOL      ExecCommand;
};

```

该协议的主要的功能是实现对 USB Key 设备传输功能的初始化和 USB Key 读写命令的初始化，实现对 USB Key 的 I/O 抽象。其中两个成员函数的定义如下：

```

EFI_STATUS
(*USBKey_INIT_TRANSPORT) (
    IN    EFI_USB_IO_PROTOCOL    *Usb,
    OUT   VOID                    **Context    OPTIONAL
);
EFI_STATUS
(*USBKEY_CMD_PROTOCOL) (
    IN    _USBKEY_CMD_PROTOCOL    *This,

```

```
IN  EFI_USB_DATA_DIRECTION    DataDir,  
IN  UNITN                     DataLen,  
IN  VOID                      *Data,  
  
);
```

其函数功能的实现过程如下：

a) 利用 USB I/O 协议实现对 USB 设备传输功能的初始化。

b) 利用 USBKEY_CMD_PROTOCOL()函数实现对 USB Key 的 I/O 抽象，此函数有四个入口函数，This 为指向 USBKEY_CMD_PROTOCOL 的指针；DataLen 表示此次传输数据的长度，Data 指向需要传输的数据；DataDir 则用于区分数据的传输方向，是输入还是输出。

其具体主要的实现流程：首先设置相关参数，如传输的数据及数据大小等；其次，利用 USB I/O 协议提供的异步传输服务进行中断传输，并通过遍历所有的 Endpoint，找出用于中断传输的 Endpoint 地址；最后，通过检查 DataDir 参数，区分其传输方向，开始相应数据的传输。

c)通过 InstallProtocolInterface 引导服务将 USBKEY_CMD_PROTOCOL 安装到相应的设备句柄上，从而实现 USB Key 驱动程序的 I/O 抽象。

3、USBKeyDriverBindingStop

USBKeyDriverBindingStop 函数基本是 USBKeyDriverBindingStart 过程的逆过程，需要依次关闭 Start 过程中打开的协议并释放 USBKeyDriverBindingStart 中所申请的空间。其函数定义如图 20 所示。

```
EFI_STATUS  
EFIAPI  
USBKeyDriverBindingStop (  
IN  EFI_DRIVER_BINDING_PROTOCOL  *This,  
IN  EFI_HANDLE                   Controller,  
IN  UINTN                        NumberOfChildren,  
IN  EFI_HANDLE                   *ChildHandleBuffer  
);
```

图 20 USBKeyDriverBindingStop 定义

其中，*This 是指向 EFI_DRIVER_BINDING_PROTOCOL 实例的指针；Controller 是传入的 USB Key 设备句柄；NumberOfChildren 和*ChildHandleBuffer 分别是设备下的子句柄和指向子句柄的指针，因为在 UEFI 规范中设备驱动不需要为设备创建子句柄，因此，这两个参数在本驱动程序中都被忽略。

其具体实现过程如下：

1) 利用 UnstallProtocolInterface 服务将 USBKEY_CMD_PROTOCOL 从 USB Key 设备句柄上卸载。

2) 通过引导服务中的 CloseProtocol 关闭在设备句柄上的打开的 EFI_USB_I/O_PROTOCOL 协议，并释放之前所申请的内存空间。

自此，实现了 USB Key 驱动程序中的 USBKEY_DRIVER_BINDING_PROTOCOL 程序。

4.3.2.1 性能优化

在上述驱动程序的实现过程中，其传输过程采用的是中断传输，其特点是反应快，但是传输数据很少，主要是用在鼠标和键盘这种设备中，相对而言 USB Key 设备与主机的通信数据比较大。考虑到在 UEFI 系统中存在 USB 大容量存储设备驱动，并在其中实现了中断和控制传输过程。因此，为了减少系统代码量，提高系统的运行效率，在 USB Key 驱动程序中可以利用中断传输和控制传输两种模式共同实现 USB Key 和主机的通信过程，其具体的实现过程与上述过程基本一致，此处不早赘述。

4.3.1 其它相关协议的实现

1、USBKeyEntryPiont

USBKeyEntryPiont 是驱动程序的入口函数，当系统调用 StartImage 系统服务时，该程序取得系统的控制权。其实现方法是利用 UEFI 中的 EfiLibInstallDriverBindingComponentName2() 在驱动程序的句柄上安装 EFI_DRIVER_BINDING_PROTOCOL 与 EFI_COMPONENT_NAME_PROTOCOL。

2、EFI_COMPONENT_NAME_PROTOCOL 协议

利用该协议实现用自定义的标识符（如“USB Key Driver”）表示该驱动程序。其函数定义如图 21 所示。

```
GLOBAL_REMOVE_IF_UNREFERENCED
EFI_COMPONENT_NAME_PROTOCOL gUsbKeyComponentName = {
    UsbKeyGetDriverName,
    UsbKeyGetControllerName,
    "eng"
};
```

图 21 EFI_COMPONENT_NAME_PROTOCOL 定义

其实现过程主要依赖于 UsbKeyComponentNameGetDriverName() 和

UsbKeyComponentNameGetControllerName() 这两个函数的实现。其中 UsbKeyComponentNameGetDriverName() 主要负责从存储驱动程序名的静态表中获得该驱动程序名称；UsbKeyComponentNameGetControllerName() 通过利用引导服务中的 OpenProtocol 服务在设备句柄上打开 USBKeyDriverBindingProtocol 协议，由此获得驱动程序所管理的设备名。下面分析这两个函数的实现过程。

● UsbKeyGetDriverName

其函数定义如图 22 所示。

```
UsbKeyGetDriverName (
    IN EFI_COMPONENT_NAME_PROTOCOL *This,
    IN CHAR8                        *Language,
    OUT CHAR16                      **DriverName
);
```

图 22 USBKeyGetDriverName 定义

其中，输入参数分别为指向 EFI_COMPONENT_NAME_PROTOCOL 实例的指针和规定的语言种类，输出参数为驱动名。其实现过程是：EFI Driver Library 中存储着一系列的静态表，这些表中存储着所有的驱动名，首先在 EFI Driver Library 中建立一个静态表，该表存储着相应的设备驱动名，如图 23 所示：

```
GLOBAL_REMOVE_IF_UNREFERENCED
EFI_UNICODE_STRING_TABLE
mUsbKeyDriverNameTable[] = {
    {"eng;en", L"Usb Key Driver"},
    {NULL, NULL}
};
```

图 23 静态表

该表中的第一个参数规定了驱动名字所使用的语言，第二个参数则代表着具体的驱动名。之后在 UsbKeyGetDriverName 函数中利用 LookupUnicodeString2 函数查询 EFI Driver Library，从而得到 USB Key 驱动的静态表，并返回 USB Key 的驱动名。

● UsbKeyGetControllerName

其函数体的定义如图 24 所示。


```
EFI_STATUS
EFI_API
UsbKeyGetControllerName (
    IN EFI_COMPONENT_NAME_PROTOCOL    *This,
    IN EFI_HANDLE                     Controller,
    IN EFI_HANDLE                     ChildHandle OPTIONAL,
    IN CHAR8                          *Language,
    OUT CHAR16                        **ControllerName
);
```

图 24 UsbKeyGetControllerName 定义

其中,*This 是指向 EFI_COMPONENT_NAME_PROTOCOL 的指针;Controller 是 USB Key 的设备句柄;ChildHandle 在本程序中忽略;*Language 为系统所支持的语言类型;**ControllerName 为相应的设备名。其实现过程是通过 Controller 打开 USBKey_Cmd_Protocol, 之后通过 EfiLibLookupUnicodeString()函数查询 EFI_UNICODE_STRING_TABLE 获取 USB Key 设备名。

在 UEFI 的规范中, EFI_COMPONENT_NAME2_PROTOCOL 协议同样也可以用自定义的标识符(如“USB Key Driver”)表示该驱动程序,它与 EFI_COMPONENT_NAME_PROTOCOL 的区别仅仅在于 SupportedLanguages 字符串中的语言代码格式不同。

4.4 本章小结

本章通过对 USB 系统结构的分析研究,结合 UEFI 规范中已定义的 USB 驱动程序栈,并以 UEFI 驱动模式为基础,设计了可信启动系统架构中的关键模块:USB Key 驱动模块,并通过实现 UEFI 驱动模式中定义的相关协议以及自定义的 USB Key 抽象 I/O 协议,实现了在 UEFI 环境中对 USB Key 设备的识别与访问,并在驱动中实现了对 USB Key 的唯一性检测,为下文实现 UEFI 环境中基于 USB Key 的完整性度量程序和身份认证程序提供驱动支持,为可信启动系统的实现奠定基础。

第五章 基于 USB Key 的完整性度量与认证

在上一章中，通过对 UEFI 环境下的 USB Key 驱动程序的实现，可以使系统在 UEFI 环境下识别和访问 USB Key 设备。但是，如果要利用 USB Key 设备完成具体的命令操作，还需要相应的实现 UEFI 环境下的 USB Key 应用程序。本章首先分析了 USB Key 的命令系统，之后，根据第三章所设计的可信启动框架的具体需求，设计并实现了 UEFI 环境下的 USB Key 的相关应用程序，实现了完整性度量和验证功能和身份认证功能。

5.1 USB Key 命令系统

因为 USB Key 是一个被动传输设备，只能在接收到主机命令之后，根据命令执行相关操作。一条指令从主机发出到 USB Key 响应可以分为四个阶段：传输管理、安全管理、命令管理和文件管理，如图 25 所示。其中，应用程序的相关功能的实现主要考虑命令管理模块。

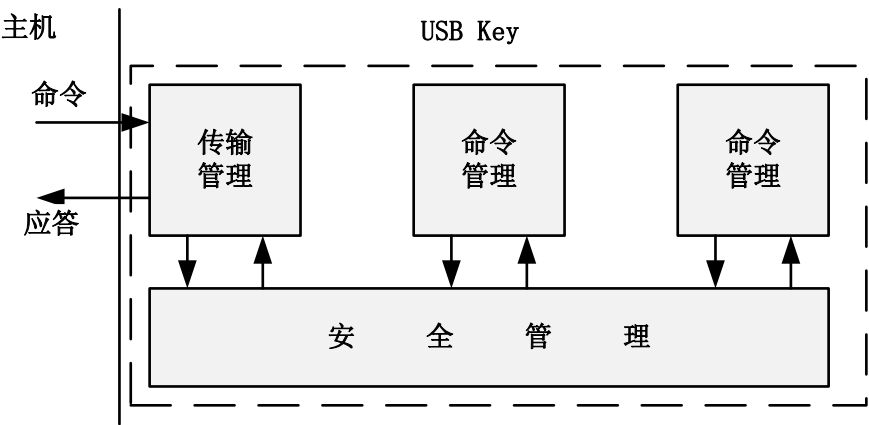


图 25 命令处理流程

命令管理模块主要实现对 USB Key 接收到的命令进行合法性检测，并在检查通过之后开始执行。下面对其命令应答机制以及命令处理流程进行相关分析，以确保应用程序的相关设计与实现符合 USB Key 的命令管理规范。

1、命令应答机制

USB Key 命令系统采用的是命令/应答的方式。这是因为 USB Key 是一个被动设备，只能根据主机发出的相关命令作出相应的应答。其命令/应答机制如图 26 所示：

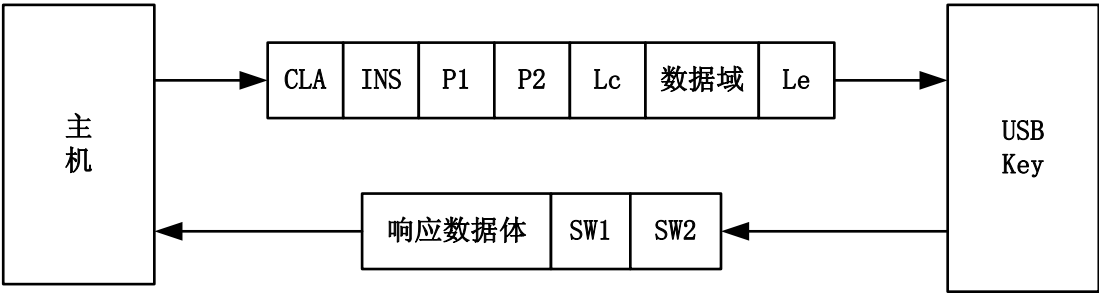


图 26 命令应答机制

在发送命令中：CLA 表示该命令的类型；INS 表示指令码；P1，P2 为命令参数；Lc 表示数据域的长度；Le 表示响应数据的长度。在响应命令中：SW1，SW2 执行命令的状态字。其中，命令数据域和响应数据体根据实际情况可能为空。

2、命令处理流程

命令处理流程指的是根据主机端发出的具体的命令参数，USB Key 需要调用不同的命令处理模块对命令进行处理，并根据处理结果返回具体的响应的过程。其命令处理流程如图 27 所示。

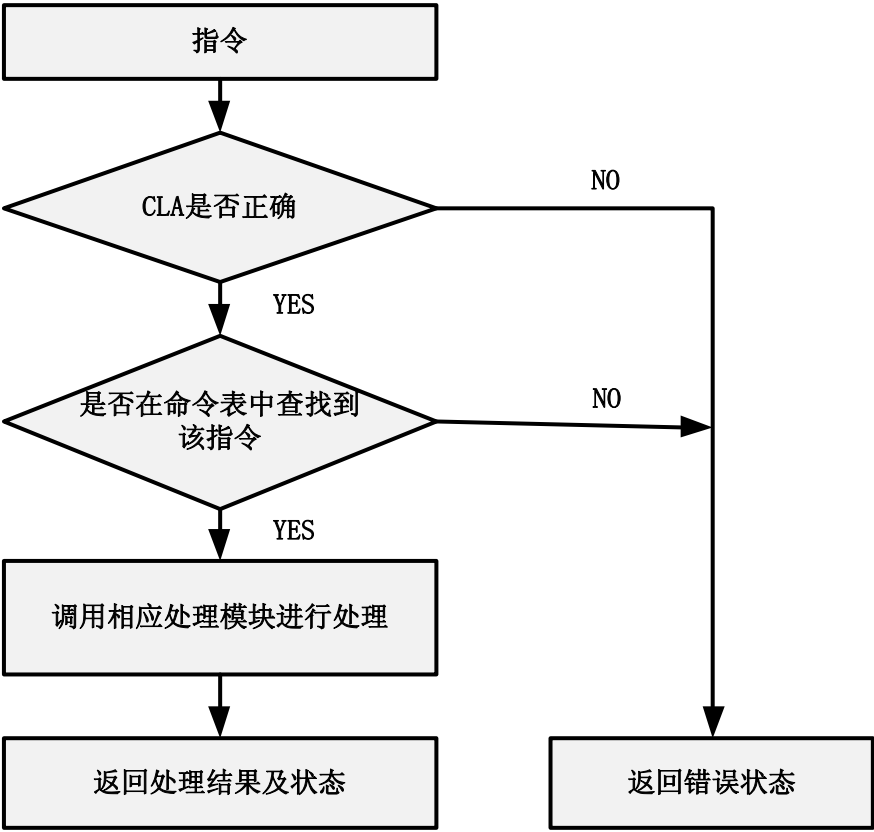


图 27 命令处理过程

其命令处理过程如下：

- 1、当 USB Key 接收到具体的指令时，首先检测 CLA 是否符合规范，验证通过则继续，否则，返回错误状态。
- 2、验证通过后，根据 INS 中的指令码在命令表中查找该命令，如果错误则返回错误状态。
- 3、如果找到该命令，则调用相应的处理模块对数据域进行处理，并通过响应数据体返回处理结果及状态。

5.2 UEFI 下 USB Key 应用程序的设计与实现

通过对 USB Key 驱动程序的实现，只是实现了系统对 USB Key 设备的访问，还需要实现相应的应用程序以实现 USB Key 具体的操作。依据系统架构其主要所需要实现的应用程序功能主要包括完整性度量功能和身份认证功能。

5.2.1 完整性度量相关程序的设计与实现

完整性度量程序主要分为两个方面：完整性度量和验证，分别用于实现对待验证组件的 hash 计算和 hash 验证操作，并且在 hash 验证操作过程中，需要实现对标准值的签名验证过程。本课题采用 sha-1 算法实现 hash 运算。该应用程序符合 UEFI 规范，可以在 UEFI 运行过程中利用 StartImage 引导服务获得系统的控制权，并运行该应用程序实现相关功能。

首先是 hash 计算操作，其入口函数符合 UEFI 规范，定义如图 28 所示

```
EFI_STATUS
EFIAPI
HashDegist (
    IN EFI_HANDLE                ImageHandle,
    IN EFI_SYSTEM_TABLE          *SystemTable
);
```

图 28 hash 计算应用程序入口函数

其中，HashDegist 函数的返回值的类型为 EFI_STATUS，为 TRUE 或者 FALSE 两种状态；ImageHandle 用来描述、访问、控制此 Image；SystemTable 是一个结构体，它是 UEFI 系统中的一个全局结构。通过该结构可以实现对 UEFI 提供的各种服务的调用。

考虑到本课题是基于 UEFI 2.3 版本的，在该版本中，系统集成 Security 模块，可以利用模块实现 sha-1 hash 运算以及 sha-1 hash 验证功能，减少系统的代码量并提高系统的运行效率。

函数体的实现过程如图 29 所示。

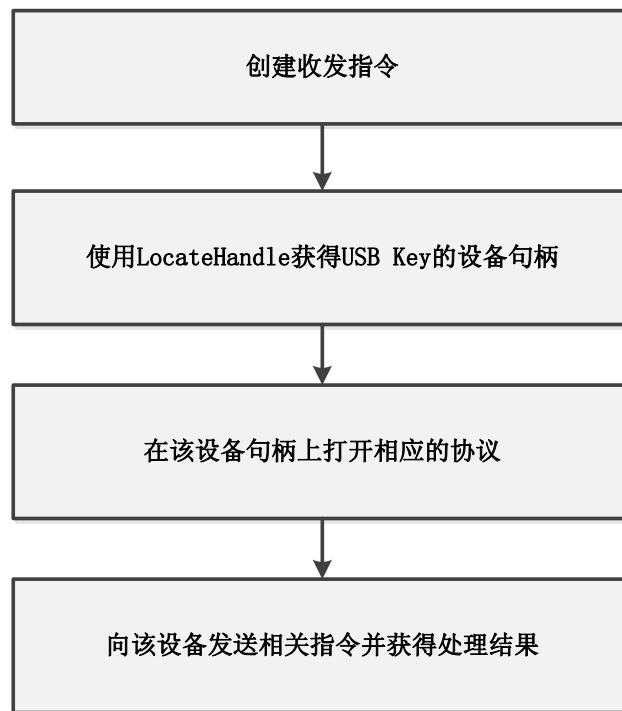


图 29 函数处理流程

具体的实现过程如下：

- 1、根据 USB Key 的指令规范，创建指令。
- 2、USB Key 完整性度量功能的实现需要依赖于 USB Key 驱动程序，因此利用引导服务 LocateHandle 函数，通过寻找 USB Key 驱动程序中的特殊协议找到 USB Key 的设备句柄。
- 3、通过在该句柄上安装 USBKEY_CMD_PROTOCOL 协议，实现 USB I/O 抽象。
- 4、填充 USBKEY_CMD_PROTOCOL 的相关参数，向 USB Key 发送 Hash 运算指令，并再次利用 USBKEY_CMD_PROTOCOL 获取其 Hash 结果。

至此，实现了 HashDegist 函数，利用该函数可以实现对待验证组件的 hash 计算的操作。

为了实现架构中度量模块的功能以及为之后可信系统的建立奠定基础，还需要实现以下几个应用函数，如表 3 所示。其实现流程与 hash 计算函数 HashDegist 的实现流程基本一致，此处不再赘述。

由表可知，完整性度量过程中需要不断调用 HashDegist 和 HashVerify 这两个函数，为了提高运行效率，避免反复调用过程。本课题在实现过程中将这两个函数进行封装成一个函数 Verify。

表 3 完整性度量需要实现的应用程序

函数名	功能
CreateRSA	生成一个 RSA 公司密钥
HashVerify	验证 Hash 值是否一致
RSAsign	实现对基准值的签名运算
RSVerify	实现签名验证运算

在前述准备工作完成后，就可以实现对引导过程的完整性度量，其过程图如图 30 所示。

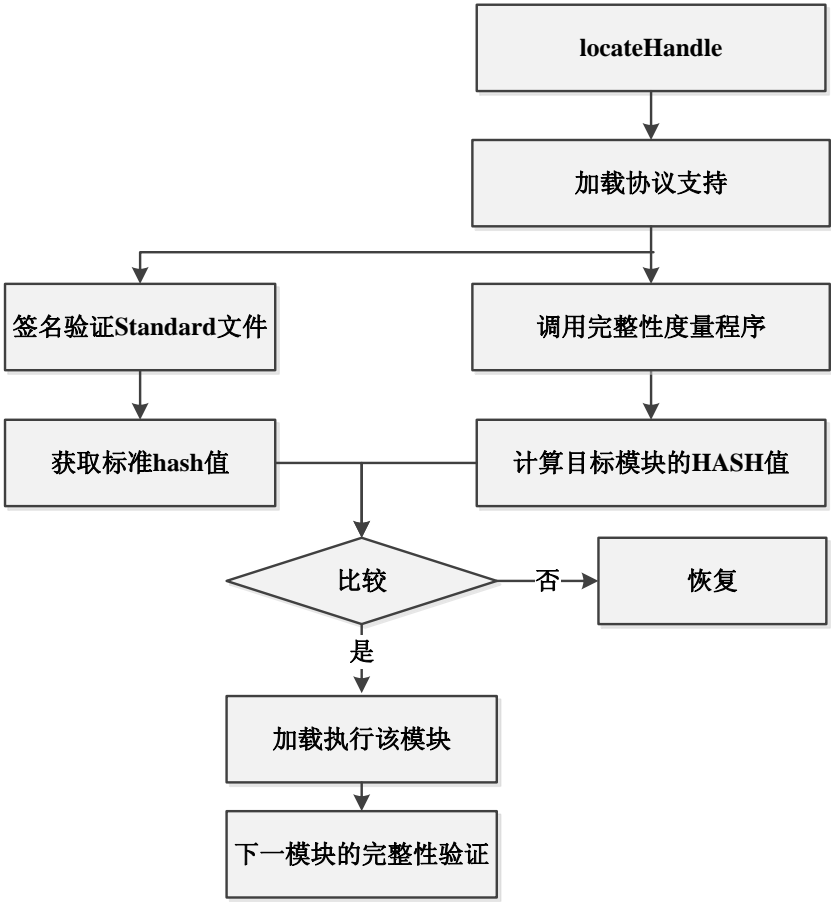


图 30 完整性度量过程

在 UEFI DXE 阶段使 USB Key 驱动先于其他驱动程序执行，在 USB Key 驱动程序完成初始化之后，首先检测 USB Key 是否存在、USB Key 的合法性及其级别（管理员还是普通用户，其用户身份的合法检测过程在 BDS 阶段，随后会详述这一过程），在前面所有验证都通过的情况下 USB Key 取得系统控制权，开始启动过程的完整性验证过程。此时，USB Key 首先利用 Boot Service 中的 LocateHandle 服务，找到 USB Key 设备所对应的句柄，据此加载 USB Key 的相应协议以实现对 USB Key 的访问，并加载和执行完整性度量程序，

实现对需要进行完整性验证的模块进行 Hash 运算，得到该模块的 Hash 值，随后利用 USB Key 自身存储的私钥对基准库中的 standard 文件进行签名验证运算，认证通过后找到对应模块的 Hash 标准值，与计算得到的 Hash 值进行比较验证，验证通过之后，USB Key 将控制权转回给系统，系统加载和执行该模块，之后继续下一模块的完整性验证过程，直到整个启动过程结束。否则，则利用恢复模块对系统进行恢复操作。

5.2.2 身份认证程序的设计与实现

身份认证功能的应用程序实现流程跟上述完整性度量程序实现的流程大同小异，都是通过其驱动程序获取 USB Key 设备句柄，并通过 USBKEY_CMD_PROTOCOL 协议实现对 USB Key 设备的访问与通信。因此，此处不再赘述其具体的实现过程，重点放在其方案的设计与实现上。

基于 USB Key 的双因子身份认证过程，相比较传统的口令验证而言，具有更高的安全性和可靠性。但由于 USB Key 设备在市场上可以很轻易的得到，这就需要对插入系统的 USB Key 的合法性进行鉴定，确保 USB Key 的唯一性。同时，为了避免系统被用户有意或无意的更改系统给系统带来潜在的安全威胁，需要对用户身份进行分级管理。对于单位用户来说，需要设立管理中心，管理中心将 USB Key 分别注册为管理员身份和普通用户身份，这就将用户分为管理员和普通用户两级，管理员可以持相应级别的 USB Key 实现对系统进行升级或者扩展，而普通用户只能实现对系统的使用和查阅。

本方案中，在 USB Key 驱动程序加载之后，首先判断是否有 USB Key 插入，在有 USB Key 插入的情况下，判断插入 USB key 的合法性和其相应的级别，主要是利用 EFI_USB_DEVICE_DESCRIPTOR 中定义的 IdVendor, IdProduct 和 BcdDevice 来区分普通用户身份和管理员身份。之后，进入 BDS 阶段，身份认证模块取得系统的控制权，并在执行完身份认证功能之后，控制权转回给系统，继续系统的可信引导工作。

用户 USB Key 级别鉴定过程如图 31 所示。

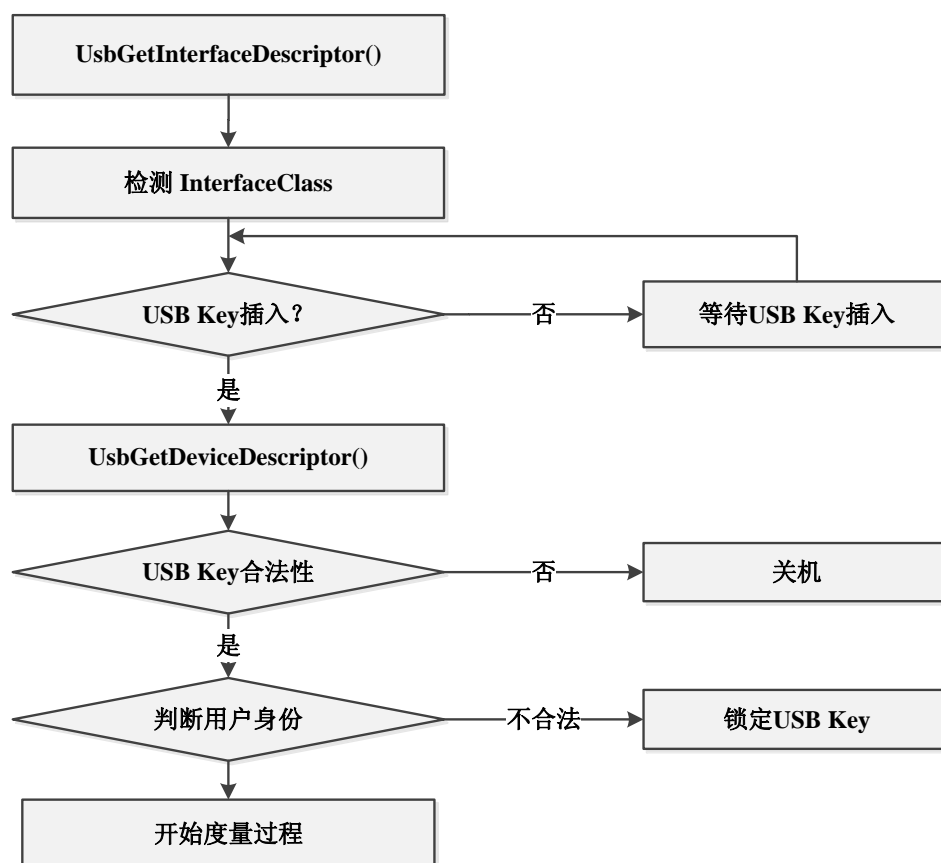


图 31 用户 USB Key 级别鉴定过程

其具体过程如下：

在 USB Key 驱动程序加载完成之后，利用 Boot Service 中的 UsbGetInterfaceService 得到 InterfaceClass 服务，验证 InterfaceClass 是否标识一个 USB Key 设备，验证通过之后利用 Boot Service 中的 UsbGetDeviceDescriptor 服务，验证其 IdVendor、IdProduct 和 BcdDevice 是否为合法 USB Key 以及该 USB Key 对应的身份级别。

之后，开始相应级别的双因子身份认证过程。每一个 USB Key 设备都具有硬件 PIN 码保护功能，其 PIN 码和硬件构成了 USB Key 认证过程的两个必要因素，即“双因子认证”。用户只能在同时获得了合法的 USB Key 和正确的 PIN 码，才可以登录系统。由于身份认证过程涉及到 PIN 码的输入，需要其他设备如键盘等的支持，因此，可以在 BDS 阶段建立身份认证模块，利用挑战应答模式实现用户身份认证过程，其过程如图 32 所示。

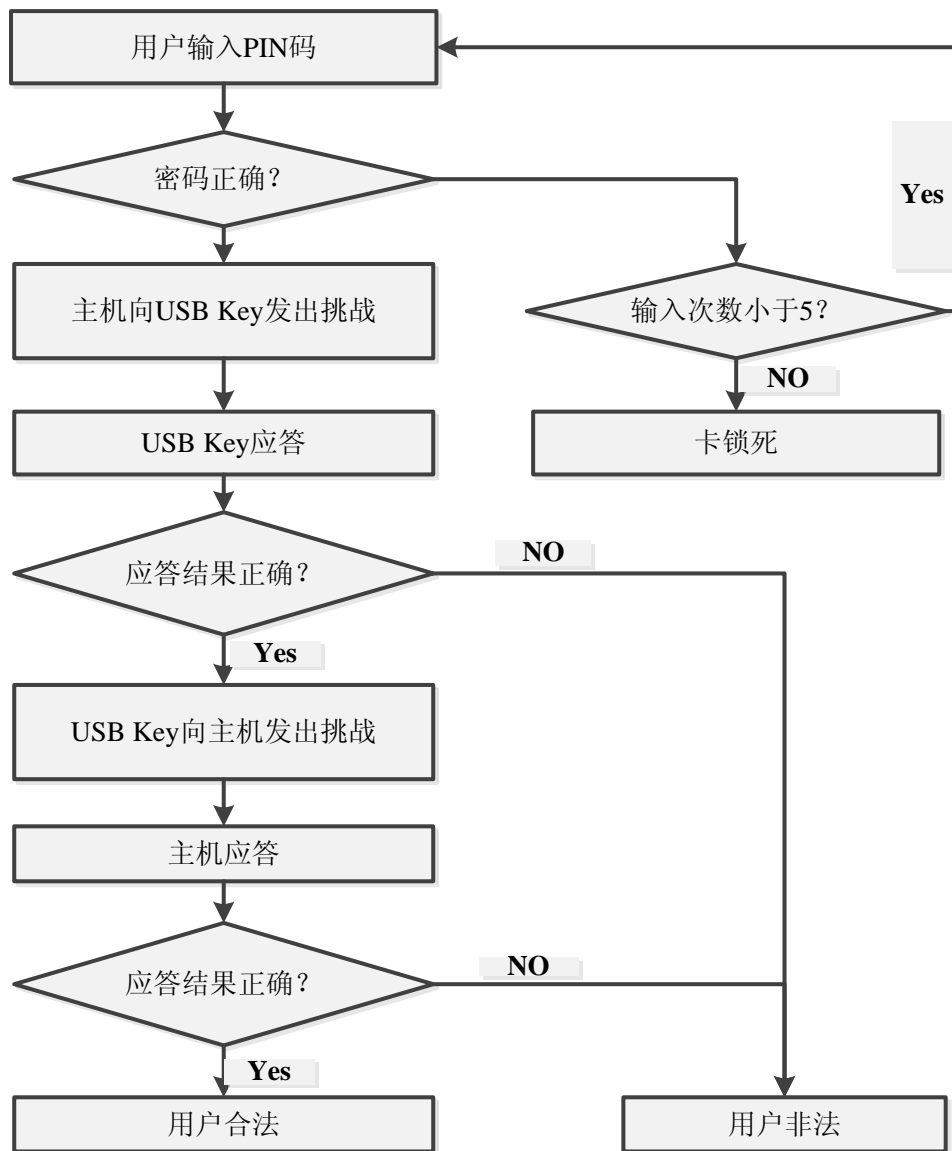


图 32 身份认证过程

在进入 BDS 阶段之后，身份认证模块获得系统的控制权，首先，在主机端生成一个随机数，并由 USB Key 利用约定好的密钥对其加密，在主机端对该密文解密得到一个数值并验证是否与生成的随机数一致。一致则主机对 USB Key 挑战成功，否则失败。同样，也需要利用 USB Key 产生一个随机数，向主机发起挑战，与上一过程类似。结果一致则表明验证成功，否则返回失败并关机。

随后，提示用户输入 PIN 码，PIN 码验证成功后则进入下一步，否则提示错误，并要求用户再次输入。如果输入 5 次，其 PIN 码依旧错误，则卡被锁定，并且只能在管理中心对卡进行重置解锁。如此可大大降低非法用户破解 PIN 码的可能性。

5.3 本章小结

本章以上一章实现的驱动程序为基础，首先分析研究了 USB Key 的命令系统，其中，

重点研究了 USB Key 的命令应答机制以及命令处理流程。结合 UEFI 规范,设计实现了基于 USB Key 系统架构中的关键模块:度量模块和认证模块,在本章体现为完整性度量程序和身份认证程序。完整性度量程序通过 UEFI 服务调用 USB Key 驱动程序的相关接口,利用 sha-1 进行完整性度量和验证;身份认证程序则是利用了 USB Key 设备实现了双因子身份认证过程,并可以对用户级别进行判定,为基于 USB Key 可行启动系统的实现奠定基础。并利用 UEFI 中已有的安全协议实现对代码的优化,减少了代码量,提高了运行效率。

第六章 基于 USB Key 可信启动系统的实现与测试

依据所设计的可信启动架构，为了保证可信启动系统的可用性、完整性与健壮性，还需要实现两个辅助模块：基准库和恢复模块。本章主要分为两个部分，第一部分通过综合分析系统效率 and 安全性以及 UEFI 相关规范，实现了可信启动系统架构中的两个辅助模块，并结合驱动程序和应用程序实现了基于 USB Key 的可信启动系统。第二部分阐述了基于 USB Key 可信启动系统的测试过程，主要包括功能测试和性能测试，并与现有的可信启动系统进行了对比分析。

6.1 基于 USB key 可信启动系统的实现

在实现了 USB Key 驱动程序以及基于 USB Key 相关操作的基础上，并根据第三章设计的基于 USB Key 的可信启动架构，就可以实现基于 USB Key 的可信启动系统。为了保证系统的完整性，本节首先对系统中的两个辅助模块：基准库和恢复模块，进行设计与实现。基准库和恢复模块的安全关系到系统度量过程以及安全响应的可靠性和可信性，如果基准库和恢复模块遭受到恶意篡改，那么整个系统的可信性将会不复存在。因为基准库和恢复模块从属于不同级别用户的操作范畴，其建立过程并不一致，所以下面分别对恢复模块和基准库的建立与实现进行分析。然后，将功能进行封装，完成在启动过程中主机和 USB Key 的控制权转换过程，建立基于 USB Key 的混合信任链，最终实现基于 USB Key 的可信启动过程。

6.1.1 基于白名单机制的基准库

系统标准值的建立与安全是可信系统架构的一个关键问题。它不仅关系着整个可信系统的度量过程的可靠性，同时，也影响着可信度量过程的运行效率。如果系统标准值遭受到恶意篡改，那么整个系统的可信性将会不复存在。

目前，基准库的建立主要有两种方法：一是将系统标准值作为 USB Key 证书的一部分存放于 USB Key 中；二是将系统标准值存储于主机端，并通过可信硬件保障其安全性。

如果将系统标准值作为 USB Key 证书的一部分，那么将会不便于之后的更新和扩展。而如果将基准库建立在主机上，因为所有级别的用户在使用电脑之前都可能需要经过可信启动过程，那么就不能简单的依靠签名机制来保证其安全性，因此如果将基准库存储在主机的端则需要花费更多的精力来确保其不被破坏。考虑到本课题所采用的 USB Key 是复合了 U 盘设备的。因此，本课题将基准库存储在每一个 USB Key 的移动存储区域，并通过 USB Key 签名运算对其签名保护，确保其安全性。

再者，为了避免用户有意或无意的添加未知的硬件，对终端主机造成不可预知的安全

威胁，本课题建立了硬件白名单。硬件白名单中存储了已知的硬件名以及由硬件特征值生成的 Hash 值。当用户需要加载某个硬件的驱动时，首先检测该硬件是否在硬件白名单中，如果不存在，则提示出现未知硬件，并终止该硬件的驱动加载过程。如果存在，则继续验证其特征值是否一致。如果一致则加载该硬件驱动，如果不一致则提示出现未知硬件，并终止该硬件驱动的加载过程。其流程如图 33 所示。

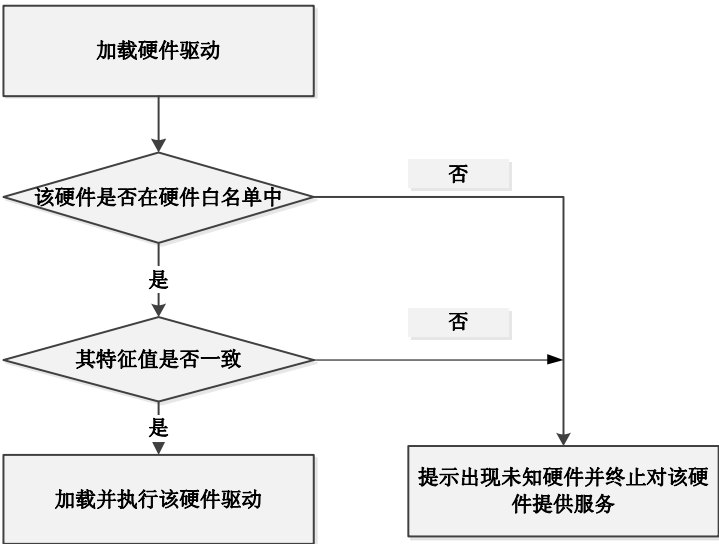


图 33 硬件白名单验证流程

为了避免系统的臃肿，简化可信系统查找硬件白名单和系统标准值的流程。因此，将系统标准值和硬件白名单共同存放在一个通过 USB Key 签名的文件内，命名为 standard 文件。standard 文件是一个 N 行两列的表，存储着目标名和相应的 Hash 标准值，部分标准值如表 4 所示。

表 4 standard 基准表

目标文件名	Hash 标准值
UsbKbDxe.efi	860790ae35505615e706fd9224f5a6f2c17dfe23
UsbMassStorageDxe.efi	cdf663858e6854c3ecb70f5f65a3552fc3816990
...	...
MBR	4379a3d43019b46fa357f7dd6a53b45a3ca8fb79
bootmgr	7a2ac87b31aa40a1ea92eb34410305fac9f8bc6a
Winload.exe	696622d25862e9868879bd9fa47d168f97d073cd
...	...

standard 文件需要通过 USB Key 进行签名保护，主要利用上一章的所实现的应用程序 CreateRSA、RSAsign 和 RSAverity 这三个函数实现。其过程是：

首先，利用 USB Key 的应用程序 CreateRSA 产生一对用于文件签名和验证的公私密钥，且私钥存储在卡内，而这个私钥是不能被取出的，这就保证了私钥的安全性。其次，利用

USB Key 计算出需要度量模块的标准值，将其存储在 standard 文件内，并利用 USB Key 的应用程序 RSASign 对 standard 文件进行签名保护。在系统进行完整性度量的过程中，首先，利用存储于 USB Key 自身的私钥对 standard 文件进行签名验证，在验证通过的情况下，取出存储在 standard 文件中的标准值。之后进行完整性验证过程。

6.1.2 基于 Capsule 更新机制的恢复模块

恢复模块存放的是受信任的启动过程中所用到的组件，它是在系统度量过程中出现不可预知的错误或者受到恶意攻击时被触发，实现对该问题组件的恢复。并且，恢复模块只能在管理员状态下被调用。目前，市场上出现了带有 U 盘的 USB Key 设备，因此，可以将恢复模块存储在管理员级的 USB Key 设备的 U 盘中。

当一个组件没有通过系统的完整性度量过程时，普通用户级的 USB Key 会中断计算机的启动过程，并发出报警。这时需要管理员级的 USB Key 实现对计算机系统的恢复，使其受损坏组件恢复正常。其恢复过程利用了 UEFI 中的 Capsule 更新机制。

Capsule 是 UEFI 定义的一种用于操作系统和固件系统之间进行数据传输的规范。它的结构如图 34 所示：

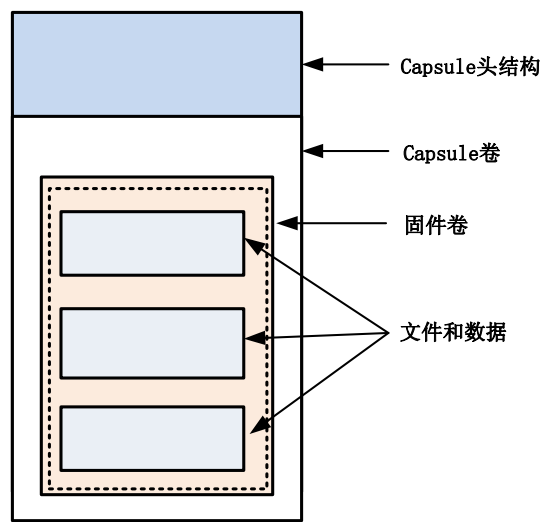


图 34 Capsule 结构^[53]

其中，Capsule 头结构包含着整个 Capsule 的相关信息，其结构体以及各参数的意义如图 35 所示。Capsule 卷中可存储固件文件、UEFI 相关程序以及 BIOS 配置信息等数据。

其系统恢复实现过程如下：

更新操作主要是在 S3 睡眠状态下执行。在 Capsule 中封装了需要更新的文件，当更新的文件较大时，在更新过程中可以按照实际情况把 Capsule 分成更小的 SubCapsule 数据单元。在 PEI 阶段，Capsule 由 Capsule PEIM 处理。Capsule PEIM 的职责是利用存储在 UEFI 运行时服务中的数据结构定位 Capsule。之后，Capsule PEIM 将可能不连续的 Capsule 各个部分放入一个连续的内存块中，并且不会跟 DXE 和 PEI 阶段的其他块产生碰撞。它通过

HOB(Hand-Off Block)来描述这个块的位置和大小。DXE 阶段负责初始化和驱使从 Capsule 卷 HOB 中提取出 Capsule 的过程。使数据转化为能够被 BDS 阶段和随后的各个阶段使用的固件卷。并根据启动顺序管理文件以及驱动依赖表达式，实现以正确的顺序执行 UEFI 的系统更新操作。

```
Typedef struct {
    ...
    UINT32 CapsuleImageSize;           //文件的大小
    UINT32 SequenceNumber;             //SubCapsule号
    EFI_GUID InstanceId;               //SubCapsule的个数
    ...
    UINT32 OffsetToCapsuleBody;        //Capsule在固件卷的位置
    ...
}EFI_CAPSULE_HEADER;
```

图 35 Capsule 头结构

考虑到 UEFI BIOS 的更新过程需要在 S3 睡眠状态下执行。为了避免非法用户通过利用 UEFI BIOS 的更新机制对系统进行非法更新或重写，进而绕过可信启动系统的完整性度量功能和用户认证功能，从而使基于 USB Key 的可信启动系统无效，实现对系统的劫持。因此，系统必须对进入 S3 睡眠状态进行管制，只有管理员才能进入 S3 睡眠状态，也就是说用户只有同时拥有管理员 USB Key 和管理员 PIN 码的情况下，才能实现对系统的更新或重写，这进一步保证了系统的安全和可信。

6.1.3 系统实现

为了防止在系统过程中反复的调用 USB Key 应用程序，提高系统的运行效率。在系统实现之前，首先对第四章中所实现应用程序进行封装，并最后实现一个自定义的安全协议 UEFI_SEC_PROTOCL，并通过该安全协议实现系统的可信启动过程。

该协议中所用到的接口函数如表 5 所示。

表 5 UEFI_SEC_PROTOCOL

成员函数	功能
CreateRSA	生成一个 RSA 公私密钥
Verify	实现完整性度量功能
RSASign	实现对基准值的签名运算
RSASVerify	实现签名验证运算
IDauth	实现系统的身份认证功能

其中，Verify 函数由第四章已经实现的 HashDegist 和 HashVerify 两个函数组成，分别

实现 hash 运算和 hash 验证操作。其输入参数包括：指向待验证文件的指针*FilePoint；文件名 FileName 和指向基准值的指针*StPoint。输出参数为度量结果*Result，为 TURE 或者 FALSE。

在前面的准备工作完成的基础上，在系统启动过程中的恰当位置调用安全协议和身份认证功能函数，就可以实现实现基于 USB Key 的可信启动系统。

可信启动系统运行的流程是：

在 UEFI 系统执行阶段，当 USB Key 设备完成初始化后，首先实现对启动顺序管理文件(a priori file)进行完整性验证，保证启动过程中驱动程序的启动顺序是按照预期执行的。之后，开始 DXE 阶段完整性度量过程。UEFI 在调用驱动程序，系统镜像或者应用程序都需要利用 LoadImage 函数，因此，可以在 LoadImage 函数中插入相关代码实现对安全协议中度量函数的调用，并实现完整性度量和验证过程，如图 36 所示。

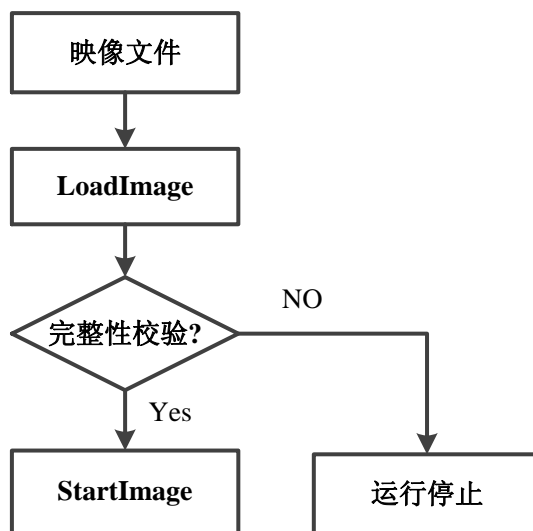


图 36 完整性度量和验证过程

其实现过程如下：

1、当一个驱动程序或者应用程序需要加载执行时，首先调用 LoadImage 函数，通过 LoadImage 函数获得 Verify 函数所需要的三个参数。

2、调用安全协议中的 Verify 函数实现对目标组件的完整性度量和验证过程，如果通过完整性验证，则继续调用 StartImage 过程开始目标组件的执行过程。如果没有通过完整性验证过程，则系统启动过程挂起，并提示出现错误的组件，提醒管理员对该组件进行恢复。

当系统在 DXE 阶段发现硬件时，首先利用其已经通过完整性度量的驱动程序收集其硬件信息，主要是接口相关信息，并对该信息进行 hash 运算。之后，搜索基准库中的硬件白名单，检测该硬件是否为可知硬件，如果存在，则将前面计算的 hash 值与白名单中的 hash 标准值相比较，两者一致，则说明该硬件是已知的。如果该硬件不在白名单中，或者

hash 值不一致，则说明该硬件为未知硬件，系统终止该硬件设备，并提示出现未知硬件。

当系统进入到 BDS 阶段后，首先进行身份认证功能。认证过程主要包括主机对 USB Key 设备的认证以及对用户身份的合法性认证。用户身份的合法性认证包括用户级别认证以及相应的 PIN 码检测两个阶段。其具体的实现过程已经在第四章中阐述，此处不再重复说明。在确定用户身份合法的情况下，继续对该阶段的应用程序以及 UEFI shell 等文件进行完整性度量和验证过程，如果通过完整性验证则继续进行下一个阶段。如果没有通过完整性验证过程，则系统启动过程挂起，并提示出现错误的组件，提醒管理员对该组件进行恢复。

在操作系统引导阶段，根据本课题提出的混合信任链式过程，在 UEFI 系统正常运行到 BDS 阶段结束时，首先完成对之后的操作系统引导和加载过程的完整性验证，只有在全部通过的情况下才能开始操作系统的引导和加载过程。如果没有通过完整性验证过程，则系统启动过程挂起，并提示出现错误的组件，提醒管理员对该组件进行恢复。

6.2 系统测试

本节对上述实现的基于 USB Key 的可信启动系统进行测试，测试主要包含功能测试和性能测试两个方面，功能测试主要实现对基于 USB Key 的驱动程序、完整性度量和验证功能以及身份认证功能的有效性进行测试，而性能测试主要是对可信系统的时间开销进行测试，最后将本系统与已有的可信启动系统进行对比分析。

首先给出整个测试环境，分别如表 6 所示。

表 6 测试环境

类别	型号
处理器	Intel Core i7 4470 @3.4GHz
主板	华硕 B85-PLUS
内存	16GB DDR3 1600MHz
硬盘	500GB 7200
显卡	Intel HD Graphics 4600
USB Key	飞天诚信 StorePass2000_FT11
UEFI	2.3
操作系统	Windows7

6.2.1 功能测试

在功能测试中，课题通过在 UEFI shell 中手动加载各个功能模块，对各个模块分开进行测试，这样能够更直观的展现本系统各个子功能的有效性。

1、驱动程序的测试

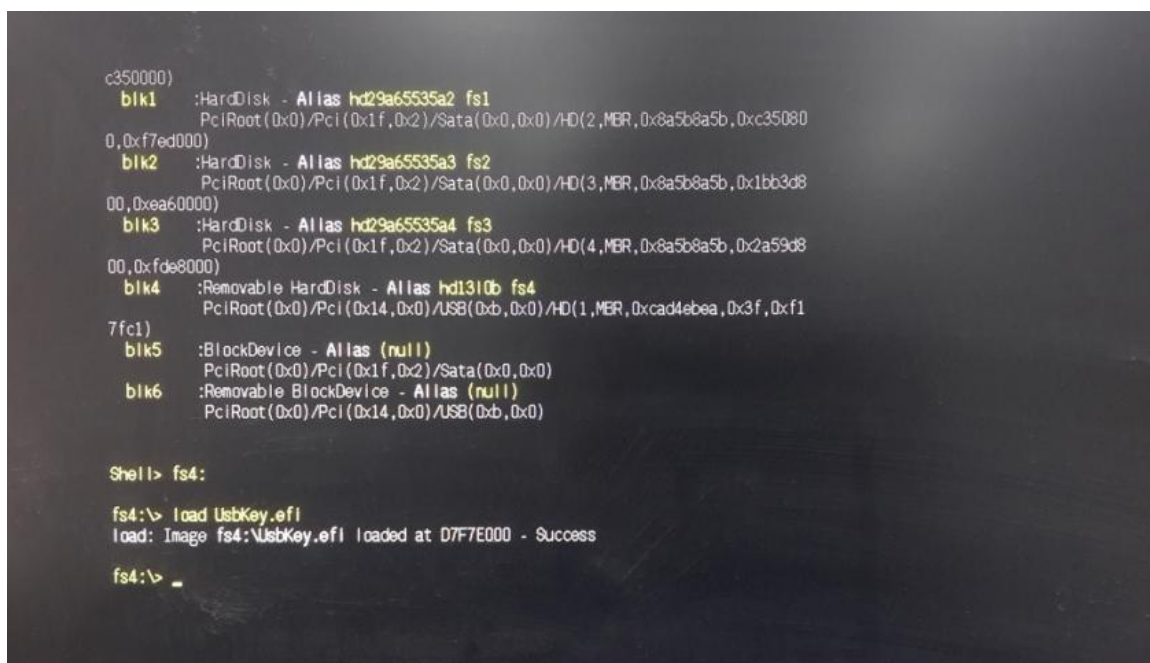
这部分测试主要是验证驱动程序是否符合 UEFI 驱动模式，能否加载运行和管理 USB Key 设备。

测试步骤：

1) 制作 UEFI shell USB 启动盘。因为目前 UEFI 只支持 FAT 文件系统，所以首先将 U 盘格式化为 FAT 格式，之后，将 shell.efi 以及所要用的测试程序，如 USB Key 驱动程序、完整性度量程序和认证程序存储到该 U 盘中。

2) 电脑加电启动，按“F2”进入系统 UEFI 界面，通过 U 盘进入到 shell，选择 U 盘对应的盘符进入 U 盘。

3) 利用 UEFI shell 命令开始系统测试。输入“load USBKey.efi”，加载驱动程序，系统显示加载成功，如图 37 所示。



```
c350000)
blk1 :HardDisk - Alias hd29a65535a2 fs1
      PciRoot(0x0)/Pci(0x1f,0x2)/Sata(0x0,0x0)/HD(2,MBR,0x8a5b8a5b,0xc35080
0,0xf7ed000)
blk2 :HardDisk - Alias hd29a65535a3 fs2
      PciRoot(0x0)/Pci(0x1f,0x2)/Sata(0x0,0x0)/HD(3,MBR,0x8a5b8a5b,0x1bb3d8
00,0xea60000)
blk3 :HardDisk - Alias hd29a65535a4 fs3
      PciRoot(0x0)/Pci(0x1f,0x2)/Sata(0x0,0x0)/HD(4,MBR,0x8a5b8a5b,0x2a59d8
00,0xfde8000)
blk4 :Removable HardDisk - Alias hd1310b fs4
      PciRoot(0x0)/Pci(0x14,0x0)/USB(0xb,0x0)/HD(1,MBR,0xcad4e8ea,0x3f,0xf1
7fc1)
blk5 :BlockDevice - Alias (null)
      PciRoot(0x0)/Pci(0x1f,0x2)/Sata(0x0,0x0)
blk6 :Removable BlockDevice - Alias (null)
      PciRoot(0x0)/Pci(0x14,0x0)/USB(0xb,0x0)

Shell> fs4:

fs4:\> load UsbKey.efi
load: Image fs4:\UsbKey.efi loaded at D7F7E000 - Success

fs4:\> _
```

图 37 USB Key 驱动程序加载测试

4) 输入“drivers”指令，该命令用于显示系统所有的驱动程序，系统显示的所有驱动程序中包含“USB Key Driver”，如图 38 所示。然后，再输入“devices”指令，该命令用户显示当前系统中连接的硬件设备，可以看到系统可以识别 USB Key 设备。最后，通过“connect”指令，实现 USB Key 驱动对 USB Key 设备的管理，系统显示连接成功。

```

C8 0000000A ? - - - - IP4 Network Service Driver      Ip4Dxe
CD 0000000A ? - - - - MTFTP4 Network Service      Mtftp4Dxe
CE 0000000A ? - - - - UDP Network Service Driver      Udp4Dxe
CF 0000000A ? - - - - DHCP6 Protocol Driver      Dhcp6Dxe
D0 0000000A ? - - - - IP6 Network Service Driver      Ip6Dxe
D1 0000000A ? - - - - MTFTP6 Network Service Driver      Mtftp6Dxe
D2 0000000A ? - - - - UDP6 Network Service Driver      Udp6Dxe
10D 0000003A D - - 3 - AMI USB Driver      UHCD
10F 0000003A B - - 3 5 USB bus      UHCD
110 00000001 D - - 1 - USB Hid driver      UHCD
111 00000001 D - - 1 - USB Mass Storage driver      UHCD
112 00000001 ? - - - - AMI USB CCID driver      UHCD
125 00000010 ? - - - - <UNKNOWN>      BIOSBLKIO
126 00000024 B - - 1 1 BIOS[INT10] Video Driver      CsmVideo
127 00000010 ? - - - - <UNKNOWN>      <UNKNOWN>
13D 00000010 D - - 7 - <UNKNOWN>      CORE_DXE
13E 00000010 D - - 1 - <UNKNOWN>      CORE_DXE
13F 00000010 B - - 4 4 <UNKNOWN>      CORE_DXE
141 00000010 B - - 2 5 <UNKNOWN>      CORE_DXE
142 00000010 D - - 2 - AMI PS/2 Driver      CORE_DXE
143 00000001 ? - - - - AMI IDE BUS Driver      CORE_DXE
146 00000010 ? - - - - <UNKNOWN>      CORE_DXE
183 0000000A ? - - - - Usb Key Driver      \UsbKey.efi

fs4:\>

```

图 38 “drivers”指令测试

该项测试说明了 USB Key 驱动程序符合 UEFI 驱动模式，可以正常运行并能实现对 USB Key 设备的设别与管理。

2、完整性度量功能测试

该部分测试主要是验证完整性度量过程能否有效鉴别系统是否被篡改，并做出相应的响应。为了更加突出显示系统的完整性度量功能的有效性，选择在不影响系统启动的情况下，手动修改计算机终端的部分启动文件，并对该部分文件进行测试。

测试步骤：

1) 与驱动测试一样，首先进入 shell 平台。

2) 首先测试完整性度量程序对 UEFI 系统下的驱动程序和应用程序的有效性。为了不影响系统的启动过程，选择以应用程序 HelloWorld.efi 为测试对象，事先在基准库中存储其标准值，之后手动修改该应用程序。然后，运行完整性度量模块对该程序进行完整性度量和验证。系统显示：驱动程序完整性验证不通过，并提醒管理员对系统进行恢复，如图 39 所示。

```

fs0:\> Verify.efi
Verify HelloWorld.efi...
sha-1:98c33819ead660e5208a001e24eee869412bd871
standard sha-1:361ad77b499d34ec0ab8fd89be1df169598004b9
Wrong!!! Call the administrator to recover the module

```

图 39 完整性测试

3) 其次测试完整性度量程序对操作系统引导阶段的有效性。操作系统引导阶段任何组件被修改都会给系统的安全造成巨大的威胁。测试中，以 MBR、Bootmgr、winload.exe

和内核加载过程中加载的 `ntoskrnl.exe` 等为样本，并逐一修改以测试系统的启动情况。其测试结果如表 7 所示。

表 7 测试结果

修改对象	测试结果
MBR	启动失败，并提示“MBR 出现错误，请管理员恢复”
Bootmgr	启动失败，并提示“Bootmgr 出现错误，请管理员恢复”
Winload.exe	启动失败，并提示“Winload.exe 出现错误，请管理员恢复”
ntoskrnl.exe	启动失败，并提示“ntoskrnl.exe 出现错误，请管理员恢复”

可以看出，系统能够发现对 MBR、Bootmgr、winload.exe 和内核加载过程的修改，并且能够终止系统的启动过程，提醒管理员对相应组件进行恢复。

测试表明，在信任链中任何一个组件被篡改时，系统都能凭借完整性度量和验证结果有效地检测到篡改行为，并且计算机终端的启动过程将会在第一时间被终止并提醒管理员对相应被篡改组件进行恢复，阻止了因此可能造成的损失，保障了系统启动过程的安全。

3、硬件白名单机制的有效性测试

该部分的测试主要用于检测终端系统在启动过程中其所初始化的硬件都是已知的，防止用户有意或无意向终端添加未知硬件而对终端主机造成威胁。其硬件白名单建立在基准库中。

测试步骤：

- 1) 进入 UEFI shell，运行完整性度量模块；
- 2) 向系统插入一个不在白名单中的 USB 设备，系统显示“未知设备，设备未初始化”，其实验结果类似完整性度量过程，因此不再截图赘述。

4、身份认证功能测试

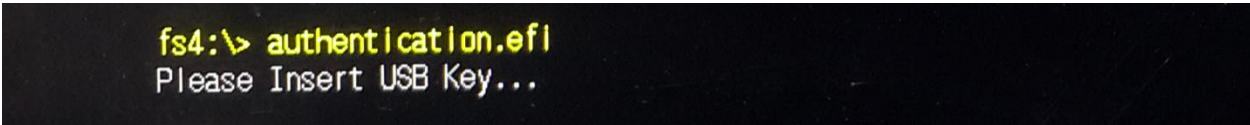
该部分测试主要验证身份认证模块能否鉴别 USB Key 的合法性和用户的合法性。

测试步骤：

- 1) 进入 UEFI shell，运行认证模块；
- 2) 首先不插入 USB Key 设备，查看是否提示未插入 USB Key；
- 3) 分别测试插入合法 USB Key 设备和非法 USB Key 设备，以测试系统是否能够验证 USB Key 的唯一性；
- 4) 分别输入错误的 PIN 码和正确的 PIN 码，已验证用户的合法性。

测试结果：

在系统启动到 BDS 阶段后，身份认证系统取得系统的控制权，提醒用户插入 USB Key。若没有 USB Key 插入，则系统一直等待，如图 40 所示。

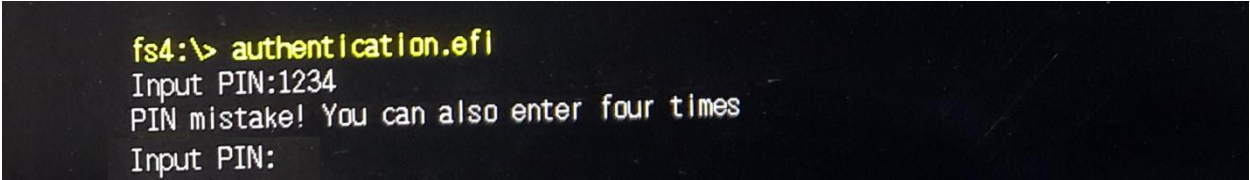


```
fs4:\> authentication.efi
Please Insert USB Key...
```

图 40 未插入 USB Key

若非法 USB Key 插入，USB Key 验证失败，直接关闭计算机。若有合法的 USB Key 插入，则提醒输入 PIN 码。

随机输入一个 PIN 码，系统提示输入错误，并要求再次输入，如图 41 所示。在连续 5 次输入错误的 PIN 码的情况下，USB Key 会被锁定，并且系统关机，USB Key 被锁定后只能通过管理员对其进行解锁操作。输入正确的 PIN 码，则系统正常开机，其实验结果不再赘述。



```
fs4:\> authentication.efi
Input PIN:1234
PIN mistake! You can also enter four times
Input PIN:
```

图 41 输入错误 PIN 码

该部分的测试表明了，身份认证功能模块能够有效验证 USB Key 设备和用户身份的合法性，降低了非法用户直接登录系统的风险，能够有效的避免系统的关键数据或文件被非法用户窃取和破坏。

6.2.2 性能分析

由于系统引入了 USB Key 设备实现完整性度量过程和完整性验证过程，相比原来的启动过程必然会带来一定时间开销。其额外增加的时间消耗主要集中在对各个组件的度量过程上。

对于完整性度量过程所增加的时间消耗，可由公式 $\Delta T = \sum t(Z_i)$ ，(i=1, 2, 3, 4, 5...) 计算得出， Z_i 为各个组件在完整性度量和验证过程的时间消耗。

因为完整性度量过程主要是利用 sha-1 算法，利用 C++ 编程测试，结果显示在 3.4GHz 的主机上 sha-1 计算 1M 文件的摘要值大概需要 7.1ms。

实验环境中的 UEFI BIOS 版本其大小大概为 5M，而需要度量的主要是 UEFI 后两个阶段的启动过程，这两个阶段包含 UEFI BIOS 中的大部分驱动程序和应用程序，约为 3.2M。在操作系统引导阶段，其内核引导和加载过程所需要的文件不超过 100M，所以，由上述分析可知，系统启动过程中，因为完整性度量过程所产生的额外消耗仅为 730ms 左右。

由于 USB Key 采用 USB 2.0 的接口规范，其传输速率是 60M/s，因此，因此主机与 USB Key 通信而造成的系统消耗大约为 1.5s。

而如果系统在度量不通过的情况下需要对系统进行恢复时，其恢复过程时间消耗相对较长。经测试，其对整个 BIOS 更新时间大约为 173s。

综上所述，系统过程在加入可信启动系统而造成的系统消耗大约为 2.2s~2.3s，而系统恢复过程可能需要消耗更高的时间。这对于用户来说是可以被接受的。

6.2.3 对比分析

在 1.3.2 节中指出了当前利用 USB Key 实现可信启动过程的两种方法：第一种方法是通过在 EFI 系统中实现 USB Key 的驱动程序与应用程序，从而实现基于 USB Key 的可信启动系统，但是该方法并没有实现之后操作系统引导过程的完整性度量，同时也没有实现在没有通过完整性度量操作的情况下的响应操作；第二种方法中，以 BIOS 和 USB Key 整体为可信链的可信根，因此，该方法只是实现了操作系统引导过程的完整性度量，忽略了对 BIOS 的安全防护。同时，本系统采取混合信任链模型，相对于传统的链式信任链模型，其信任损耗较低。因为，系统的性能跟计算机的硬件环境和软件环境有很大的关系，因此，此处只对比分析了系统功能与信任损耗程度，如表 8 所示。

虽然，系统的硬件环境和软件环境有很大的不同，但是，我们也可以对系统性能进行分析对比，因为本文采用的混合信任链过程，相对于传统的单一信任链过程，减少了指令在 USB Key 和主机之间的通信次数，因此，相对于前两种方法，本系统具有较大的执行效率，系统产生的额外开销更小。

表 8 系统对比

	BIOS 防护	认证功能	系统引导过程 防护	硬件可信检 测	恢复功能	信任损耗程 度
第一种方法	√	√	×	×	×	较高
第二种方法	×	√	√	×	√	
本系统	√	√	√	√	√	

6.3 本章小结

本章通过对系统架构中的基于白名单机制的基准库和基于 Capsule 机制的恢复模块的实现，完成了系统架构中相关辅助模块，之后，结合第四章与第五章实现的系统架构中的关键模块，通过进一步的功能封装，实现了基于 USB Key 的可信启动系统，并对其功能和性能进行了测试分析，证明了本系统的有效性、可靠性。并通过与现有工作的对比分析，证明了本系统功能的完整性与相对高校性。

结束语

1、全文总结

可信启动过程是信任链过程中最为重要的一环，保障了终端系统启动过程的安全性与可靠性，为终端系统建立了最初的可信计算环境，具有十分重要的现实意义。从研究背景中可以看出当前针对计算机终端恶意软件及恶意行为数量庞大且更新速度极快，传统的方法以及越来越不能满足当前信息安全形势的需要。通过研究可信计算理论、信任链技术以及 USB Key 的相关硬件结构，利用 UEFI 可扩展性的特点，实现基于 USB Key 的可信启动系统，该系统能够确保 UEFI 及操作系统引导过程的各个组件的完整性，保证终端启动过程的可信性和可靠性，为终端建立一个最初的可信计算环境。

总结起来，本文的研究工作主要有以下几个方面：

1、设计了基于 USB Key 可信启动的系统架构以及混合信任链的可信启动流程

通过对当前安全需求分析，依据 UEFI 可扩展性和 USB Key 的特性，设计了基于 USB Key 的可信启动系统架构，并根据 UEFI 启动过程和操作系统引导过程这两个阶段的特点，提出利用混合信任链实现该系统结构。可信启动系统在 UEFI BIOS 的启动过程中采用链式的信任链结构；在操作系统引导过程中采用星形的信任链结构，以 USB Key 和 UEFI 的部分代码为可信基，一级信任一级，逐步实现系统启动过程的可信。并深入分析了该系统框架中的关键模块和辅助模块。

2、分别对系统框架中的各个关键模块和辅助模块进行了设计与实现

通过对基于 USB Key 的可信启动系统架构的进一步研究可知，该系统中各个模块主要分为关键模块和辅助模块两大类，其中，关键模块包括驱动模块、认证模块和度量模块；辅助模块主要是基准库和恢复模块。

文中通过对 UEFI 驱动模式的深入研究以及对 USB 系统结构和 UEFI USB 驱动栈的进一步分析，设计了 UEFI 环境下的 USB Key 驱动程序，并对该驱动程序中的关键程序进行了编码实现，使 USB Key 在 UEFI 环境中可以被识别与访问，实现了系统架构中的驱动模块。

之后，利用 USB Key 的命令系统，实现了对 USB Key 的具体命令操作，并进一步实现了关键模块中的认证模块和度量模块。

文中对基准库和恢复模块进行了一定讨论，设计了基于白名单的硬件可信检测机制和基于 Capsule 的恢复更新机制。对于基准库和恢复模块因为它们属于不同的用户范畴，因此，其存储位置和方式也不同，并依据 UEFI 的 Capsule 更新机制实现了系统架构中的恢复模块。

3、对整个系统架构进行实现和验证

依据混合信任链流程以及系统整个启动过程各个阶段的特点，设计了不同度量点以确保整个系统能够按照混合信任流程执行，最后实现了基于 USB Key 的可信启动系统，并对系统的功能和性能进行了测试和对比分析。测试过程主要围绕系统关键模块的有效性。测试表明本文所设计的基于 USB Key 的可信启动架构是正确而有效的。

2、工作展望

本文主要以终端系统启动过程为研究对象，根据可信计算思想以及 TPM 的在实际运用过程中的种种缺陷，设计并实现了基于 USB Key 的可信启动过程，并实验验证了其有效性。但因为从事该课题的研究时间还比较短，系统还不够完善，并且随着攻击手段的越来越丰富，系统依旧面临着众多安全挑战。下一步工作主要有：

1、系统启动过程的整体防护

因为 USB Key 为被动设备，并且需要主机端提供驱动支持，所以在本课题的系统中的可信基中包含 UEFI 前两个阶段，这就造成了系统的防护并不完整。所以在后续的工作中，需要设计全面地系统启动过程的完整性防护。

2、进一步降低系统开销

系统的额外开销主要集中于完整性度量和主机与 USB Key 的通信过程，下一步可以从优化代码和提高硬件技术来实现进一步降低系统的额外开销，增加系统的执行效率。

致 谢

时光荏苒，转眼间三年的硕士研究生活即将结束。借学位论文完成之际，衷心地向所有陪伴着我走过这段难忘岁月的人们表示感谢。

我要感谢我的导师曾光裕副教授。学术上，她那严谨的治学态度和精益求精的工作要求令我受益匪浅，也促使我不断进步。生活中，她像慈母一般的关心和爱护着实验室的每一个学生，令我们感受到家的温暖。

我要感谢实验室的李清宝教授。他那渊博的学识，从容的心态给我留下了深刻的印象，每当我看不清前进的方向时，他总能及时的给我以正确的指引。

我要感谢王炜老师，他是我这三年来的良师益友。学术上，他那丰富的科研经验总能找到解决问题的方法。生活上，每当我遭遇挫折时，他也不断的鼓励我、开导我，让我重新振作。

我要感谢实验室的陈志锋师兄，他不仅教会了我很多学习方面的知识，更教会了许多为人处世的道理。

我要感谢实验室的郭松辉、林夕杰、姜子峰、张贵民、翁晓康、王玉龙、张擂、冯培钧，他们给了我一个积极向上的学习和工作环境。

我要感谢班里的赵利军、骆凯、蒋和国、王飞龙、王奕森、杨超，谢谢你们给了我一个友爱的集体环境。

最后我还要感谢在百忙之中依然抽出时间来参加论文评阅和答辩的各位专家，是你们给了我这个总结三年学习成果的机会，请允许我向你们致敬。

贾天江
2015 年

参考文献

- [1] 2013 年中国互联网网络安全态势综述[EB/OL]. <http://www.cert.org.cn/publish/main/upload/File/CN-CERT%202013.pdf>, 2014.
- [2] Microsoft. A history of Windows [EB/OL]. <http://windows.microsoft.com/en-US/windows/history>, 2013.
- [3] Thorsten L. What's new in Linux 3.6 [EB/OL]. <http://www.h-online.com/open/features/What-is-new-in-Linux-3-6-1714690.html>, 2012.
- [4] Microsoft. A history of Windows [EB/OL]. <http://windows.microsoft.com/en-US/windows/history>, 2013.
- [5] CERT. Vulnerability Notes Database [EB/OL]. <http://www.kb.cert.org/vuls/>, 2013.
- [6] 迈克菲威胁报告: 迈克菲实验室威胁报告(2014 年 11 月刊)[EB/OL]. <http://www.mcafee.com/cn/resources/reports/rp-quarterly-threat-q3-2014.pdf>, 2014.
- [7] 徐明迪, 张焕国, 张帆, 等. 可信系统信任链研究综述[J]. 电子学报, 2014, 42(10): 2024-2031.
- [8] TCG. TPM Main Specification Part 1-Design Principles Specification, Version 1.2[EB/OL]. <http://www.trustedcomputinggroup.org>, 2007.
- [9] TCG, Design, Implementation and Usage Principles version 2.0[EB/OL]. <http://www.trustedcomputinggroup.org>, 2005.
- [10] Cooper D, Polk W, Regenscheid A, et al. BIOS Protection Guidelines [J]. NIST Special Publication, 2011.
- [11] Chao Y, Meng-ting Y. Security Bootstrap Based on Trusted Computing[C]. Networks Security Wireless Communications and Trusted Computing (NSWCTC), 2010 Second International Conference on Hangzhou. IEEE, 2010.
- [12] Gongshen L, Kui M, Siyuan F. The Research of Malware Prevention Technology Based on UEFI[C]. Proceedings of the 2012 International Conference on Electronics, Communications and Control. IEEE Computer Society, 2012.
- [13] 沈昌祥, 张焕国, 王怀民, 等. 可信计算的研究与发展. 中国科学: 信息科学, 2010, 40(2): 139-166
- [14] Wang S, Wang Y, Tian W. Research on trusted computing implementations in windows[C]. Information Science and Management Engineering (ISME), 2010 International Conference of. IEEE, 2010.
- [15] Han L, Liu J, Han Z, et al. Design and implementation of a portable TPM scheme for general-purpose trusted computing based on EFI[J]. Frontiers of Computer Science in China. 2011.
- [16] 赵波, 张焕国等. 可信 PDA 计算平台系统结构与安全机制[J]. 计算机学报, 2010, 33 (1) .
- [17] 徐震, 沈丽红, 汪丹. 一种可配置的可信引导系统[J]. 中国科学院研究生学报, 2008, 25(5): 626-630.
- [18] Junkai G, Weilong J. A Secure Bootstrap Based on Trusted Computing[A]. In: International Conference of New Trends in Information and Service Science[C]. Beijing: IEEE Computer Society, 2009: 502-504

- [19] Cole Q. Hacking Grub for Fun and Profit[EB/OL]. <http://www.pharck.org/issues.html?issue=63&id=10>, 2011.
- [20] Smith N M, Zimmer V J, Moore V C. Methods and apparatus for trusted boot optimization [Z]. Google Patents, 2011.
- [21] Mukhamedov A, Gordon A D, Ryan M. Towards a verified reference implementation of a Trusted Platform Module [M]. Security Protocols XVII, Springer, 2013,
- [22] 姚金魁, 张涛, 王金双等. 一种不依赖 TPM 的安全引导方式的设计与实现[J]. 计算机技术与发展, 2012, 22(6): 143-146.
- [23] TPM Main Design Principles Specification version 1.2[EB/OL]. <http://www.trustedcomputinggroup.org>, 2007
- [24] John M, Sean W, Omen W, et al. Experimenting with TCG/TPM Hardware, Or: How I Learned to Stop Worrying and Love The Bear[R]. Technical Report TR2003-476, Department of Computer Science, Dartmouth College, December 2003.
- [25] 张焕国, 赵波. 可信计算 [M]. 武汉: 武汉大学出版社, 2011.
- [26] Selhorst M, Stueble C. Trusted GRUB, University of Bochum [EB/OL]. <http://www.prosec.rub.de/trusted-grub.html>, 2004.
- [27] 姜政伟, 王晓箴, 刘宝旭. 基于最小攻击树的 UEFI 恶意行为检测模型[J]. Computer Engineering and Applications, 2012, 48(32).
- [28] BM. TCG GRUB [EB/OL]. <http://trousers.sourceforge.net/grub.html>, 2005.
- [29] Lin K J, Wang C Y. Using tpm to improve boot security at bios layer[C].Consumer Electronics (ICCE), 2012 IEEE International Conference on. IEEE.
- [30] Trimberger S, Moore J. FPGA security: From features to capabilities to trusted systems[C]. Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE. IEEE, 2014.
- [31] 付思源. 基于统一可扩展固件接口的恶意代码防范系统研究[D]. 上海交通大学, 2011.
- [32] 段晨辉. UEFI BIOS 安全增强机制及完整性度量的研究[D]. 北京工业大学, 2014
- [33] Cole Q. Hacking Grub for Fun and Profit[EB/OL]. <http://www.pharck.org/issues.html?issue=63&id=10>, 2011.
- [34] Bernhard K. Authenticated Booting for L4[EB/OL]. http://os.inf.tu-dresden.de/papers_ps/kauer-beleg.pdf, 2004.
- [35] Liu G, Zhang X, Zhang Y. A Method of Data Storage and Management of Embedded Trusted Platform Module[C]. 2014 International Conference on Future Computer and Communication Engineering (ICFCCE 2014). Atlantis Press, 2014.
- [36] Datta A, Franklin J, Garg D, et al. A logic of secure systems and its application to trusted computing[C]. Security and Privacy, 2009 30th IEEE Symposium on. IEEE, 2009: 221-236.

- [37] 田凯. 基于 PCI 卡的可信引导技术研究及应用[D]. 南京:南京理工大学, 2011.
- [38] 付思源, 刘功申, 李建华. 基于 UEFI 固件的恶意代码防范技术研究[J]. 计算机工程, 2012, 38(9): 117-120.
- [39] Mukhamedov A, Gordon A D, Ryan M. Towards a verified reference implementation of a Trusted Platform Module [M]. Security Protocols XVII, Springer, 2013.
- [40] 刘琛. 基于 USB-Key 的身份认证系统的设计与实现[D]. 西安电子科技大学, 2014.
- [41] Emanuele Cesena, Gianluca Ramunno, Roberto Sassu. On scalability of remote attestation. Proceedings of the sixth ACM workshop on Scalable trusted computing, 2011
- [42] Trust Computing Group. TCG Specification Architecture Overview, Version 1.4[EB/OL]. <http://www.trustedcomputinggroup.org>, 2007.
- [43] Unified Extensible Firmware Interface Specification, version 2.4[EB/OL]. [2014-07-05]. <http://www.uefi.org>
- [44] Unified EFI Forum Platform Initialization Specification Version 1.2[EB/OL]. <http://www.uefi.org>. 2011-10-17
- [45] 邢卓媛. 基于 UEFI 的网络协议栈的研究与改进[D]. 华东师范大学, 2011
- [46] Vincent Zimmer, Michal Rothman, Robert Hale. Beyond BIOS: Implementing the Unified Extensible Firmware Interface with Intel's Framework [M]. Intel Press, 2006.
- [47] 谭良, 周明天. 一种并行可恢复可信启动过程的设计与实现[J]. 计算机科学, 2007
- [48] Fang W, Yang B, Peng Z, et al. Research and Realization of Trusted Computing Platform Based on EFI[C]//Electronic Commerce and Security, 2009. ISECS'09. Second International Symposium on. IEEE, 2009, 1: 43-46
- [49] 李振华. 基于 USBKEY 的 EFI 可信引导的设计与实现[D]. 北京: 北京交通大学, 2008
- [50] 阮洪升. 基于 USBKey 的可信安全增强系统的设计与实现[D]. 长沙: 国防科技大学, 2011
- [51] Smith N M, Zimmer V J, Moore V C. Methods and apparatus for trusted boot optimization [Z]. Google Patents, 2011.
- [52] Bashun V, Sergeev A, Minchenkov V, et al. Too Young to be Secure: Analysis of UEFI Threats and Vulnerabilities [J]. 2013 14th Conference of. IEEE, 2013: 16-24.
- [53] Intel Platform Innovation Framework for EFI Architecture Specification, Version 0. 9[EB/OL]. <http://www.uefi.org>. 2003
- [54] 冯登国, 秦宇, 汪丹, 等. 可信计算技术研究[J]. 计算机研究与发展, 2011, 48(8): 1332-1349.
- [55] Zhang D, Han Z, Yan G. A portable TPM based on USB key[C]//Proceedings of the 17th ACM conference on Computer and communications security. ACM, 2010: 750-752.
- [56] Jain R, Bhatia S S, Jain D C. An Efficient Two Way Authentication Technique for the Protection of Personal Computer System[J]. International Journal of Advanced Research in Computer Science, 2012

[57] 徐明迪, 张焕国, 赵恒, 等. 可信计算平台信任链安全性分析[J]. 计算机学报, 2010.

作者简介

一、个人简历

贾天江，男，1989 年 2 月出生，河北临漳人

2012 年 6 月毕业于北京理工大学信息与电子学院信息对抗技术专业，获得学士学位

2012 年 9 月入解放军信息工程大学读硕士研究生

二、攻读硕士学位期间发表的学术论文

1. 基于 USB Key 的可信启动技术研究，《信息工程大学学报》，已录用，第一作者
2. 基于剪贴板监控的电子文档多级保护，《计算机与现代化》，已录用，第三作者

三、攻读硕士学位期间的科研情况

1. 国家核高基重大专项 0412 子项，主要参与人
2. 信息工程大学未来基金 1201，主要参与人