ICSP2008 Proceedings

Design and Implementation of UsbKey Device Driver based on Extensible Firmware Interface

TANG Weimin [1], PENG Shuanghe [1,2], HAN Zhen [1]

1 School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, Beijing, China
2 Computer Center, Beijing Information Science & Technology University, Beijing 100085, Beijing, China

twm68@sohu.com, shhpeng@bjtu.edu.cn

Abstract

The goal of trusted computing proposed by TCG is to enhance the security of platform by the way of integrity measurement. TPM is a tamper-resistant hardware module designed to provide robust security capabilities like remote attestation and sealed storage for the trusted platform. But TPM has its limitation. It cann't be directly used in common PC current in use because of its interface with PC. A UsbKey is a USB device with capabilities of smart card. Extensible Firmware Interface (abbreviated as EFI) is intended as a significantly improved replacement of the old legacy BIOS. How to design and implement the driver of UsbKey based on EFI is what this paper focuses on. It is a basement for the trusted application.

1. Introduction

A trusted computing platform is defined by the TCG as a platform that is equipped with a Trusted Platform Module (TPM) [1]. This TPM is a dedicated low cost hardware component that creates a Root of Trust. It is the foundation on which a trustworthy system can be built. TPM is embedded into the motherboard as defined by TCG. In order to enhance the security of the information by the way of trusted platform, all the PC current in use must be hardware-modified to get the feature of trusted platform. The cost is a huge problem.

In order to address this issue, how to get the trusted platform feature in current PC platform is what our lab focus on. A UsbKey device is proposed to solve this problem. The interface between the UsbKey device and the host is not LPC (Low Pin Count) bus embedded onto the motherboard, but USB bus which is a common interface in use. The UsbKey can provide the same capabilities such as integrity measurement, cryptographic key services, protected storage and endorsement services like TPM defined by TCG.

EFI is intended as a significantly improved replacement of the old legacy BIOS firmware interface historically used by all IBM PC compatible personal computers[2].EFI contains an extensibility mechanism that will allow future media devices to be supported. EFI is a new technology, there is few people have a research on the driver development at present. Since we use UsbKey as a root of trust on current PC platform, if an EFI firmware is used instead of the BIOS, how to design and implement the driver for UsbKey based on EFI is what this paper focuses on.

2. EFI Driver Model

The Extensible Firmware Interface provides a driver model for support of devices that attach to today's industry-standard buses, such as PCI and USB, and architectures of tomorrow [3]. The EFI Driver Model [4] is intended to simplify the design and implementation of device drivers and produce small executable image sizes. The file for a driver image must be loaded from some type of media. Once a driver image has been found, it can be loaded into system memory with the Boot Service LoadImage (). LoadImage () loads a PE/COFF formatted image into system memory. A handle is created for the driver, and a Loaded Image Protocol instance is placed on that handle. A handle that contains a Loaded Image Protocol instance is called an Image Handle. At this point, the driver has not been started. It is just sitting in memory waiting to be started. After a driver has been loaded with the Boot Service LoadImage(), it must be started with the Boot Service StartImage(). The entry

978-1-4244-2179-4/08/$25.00 ©2008 IEEE

point for a driver that follows the EFI Driver Model must follow some strict rules. First, it is not allowed to touch any hardware. Instead, it is only allowed to install protocol instances onto its own Image Handle. A device driver that follows the EFI Driver Model is required to install an instance of the Driver Binding Protocol onto the same image handle on which the driver was loaded. It may optionally install the Driver Configuration Protocol, the Driver Diagnostics Protocol, or the Component Name Protocol. An Image Handle that contains a Driver Binding Protocol instance is known as a Driver Image Handle. Figure 1 shows a possible configuration for the Driver Image Handle.

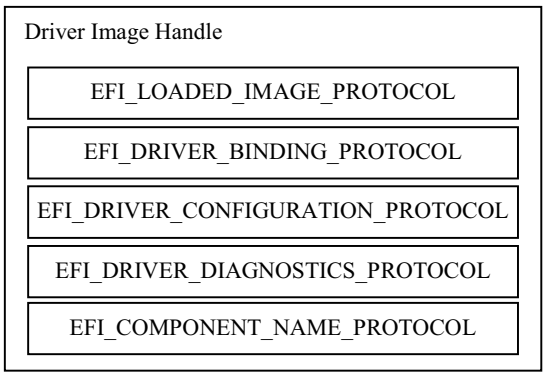

Figure 1 Driver Image Handle

It then waits for the system firmware to connect the driver to a controller. When that occurs, the device driver is responsible for producing a protocol on the controller's device handle that abstracts the I/O operations that the controller supports.

3. Hardware Components of UsbKey

Since UsbKey has the similar capabilities as TPM defined by TCG, it should provide root of trust and root of measurement for the platform. Smart cards are now widely used world wide with a microprocessor and memory embedded inside. A key feature of smart cards is that they provide secure storage for data. The smart card can store a user's key pair and an associated public key certificate. It can also complete private key operations on behalf of the user. Increasingly, smart cards are generating the key pairs automatically.

Since smartcard has the same feature as TPM, we use smartcard as the key component of the UsbKey device.

Hardware components of UsbKey are given in the following. Figure 2 shows the Functional Block Diagram of the UsbKey. The UsbKey is mainly made up of two components: USB Controller Chip and Smart Card.

(1) USB Controller Chip [5]. It is a main controller component for the UsbKey device. It is made up of four parts, i.e. USB Transceiver, Serial Interface Engine (SIE), 16 bit Micro-Processor and GPIO(general purpose I/O) Controller. The command and data channel between the Usb controller chip and host is through USB Transceiver, which supports USB2.0 protocol. The USB protocol is decoded by SIE. GPIO Controller is responsible for building information channel with smart card. Command and data is transferred bit by bit through GPIO line.

(2) Smart card Chip. Smart card [6] is the key component of the UsbKey device. Its main purpose is cryptography operations. We select a smart card compatible with protocol T=0. The T=0 protocol is an asynchronous character-oriented protocol where an acknowledgement must be received for every byte that is sent. The smart card can provide the similar capabilities as TPM.

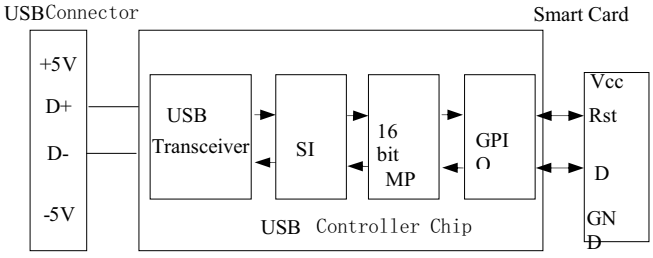

Figure 2. Functional Block Diagram of the UsbKey

4. Implementation of the UsbKey Device Driver

It is important to note that "EFI driver" in this context has no connection to OS-present drivers for Windows*, Linux*, or any other conventional OS. In the EFI Framework [7], the term "EFI driver" is used to refer to a modular piece of code that runs in the DXE phase. Since EFI is a replacement for legacy BIOS, we should communicate with the U-Key through a USB port before the operating system is loaded. At present, in the DXE phase of EFI such a driver didn't exist, so we created one.

4.1 UsbKey Device Driver Stack

In general, applications rely on the operating system's USB driver (USBD) to access the USB device; the USBD layer relies on a common interface to the hardware, which is called the host controller interface (HCI). Before the operating system is loaded we can't access the USBD, so to access the USB device during the DXE phase, we need to get to the HCI directly.

Figure 3 below shows the UsbKey device driver stack and the protocols that the UsbKey device driver consumes and produces following the EFI Driver Model. In the stack, we assume that the platform hardware produces a single USB host controller on the PCI bus. The PCI bus driver will produce a handle with EFI_DEVICE_PATH_PROTOCOL and EFI_PCI_IO_PROTOCOL installed for this USB host controller. The USB host controller driver depends on which USB host controller specification that the host controller is based. Currently, the major two types of USB host controllers are the Universal Host Controller Interface (UHCI) [8] and Open Host Controller Interface (OHCI) [9]. We selected UHCI in the prototype implementation.

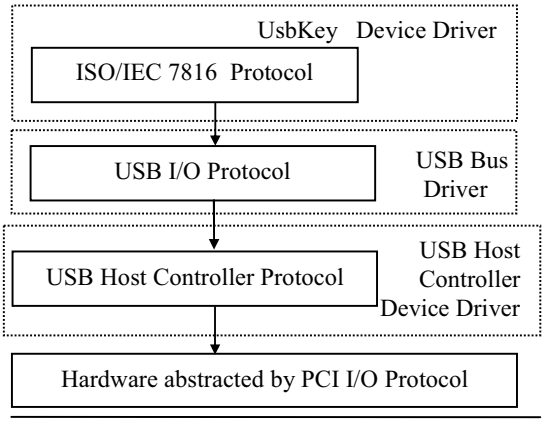

Figure 3. UsbKey Device Driver Stack

The USB host controller driver will then consume EFI_PCI_IO_PROTOCOL on that USB host controller device handle and install the EFI_USB_HC_PROTOCOL onto the same handle. The USB bus driver consumes the services of EFI_USB_HC_PROTOCOL. It uses these services to enumerate the USB bus. In the stack, the USB bus driver detected a UsbKey device. As a result, the USB bus driver will create one child handle and will install the EFI_DEVICE_PATH_PROTOCOL and EFI_USB_IO_PROTOCOL onto the handle.

The UsbKey device driver will consume EFI_USB_IO_PROTOCOL and produce EFI_SMARTCARD_IO_PROTOCOL. OS loader and other EFI shell applications can locate this EFI_SMARTCARD_IO_PROTOCOL in the handle database to use the cryptography function provided by the UsbKey device.

4.2 Implementation of UsbKey Device Driver based on EFI

According to EFI Driver Model, UsbKey device driver based on EFI is built as follows: First the UsbKey device driver needs to produce a Driver Binding Protocol. In the start ( ) service of the Driver Binding Protocol, EFI_SMARTCARD_IO_PROTOCOL is installed in the handle. Figure 4 below shows the UsbKey device handle and protocols from the handle database that is produced by the UsbKey device driver.

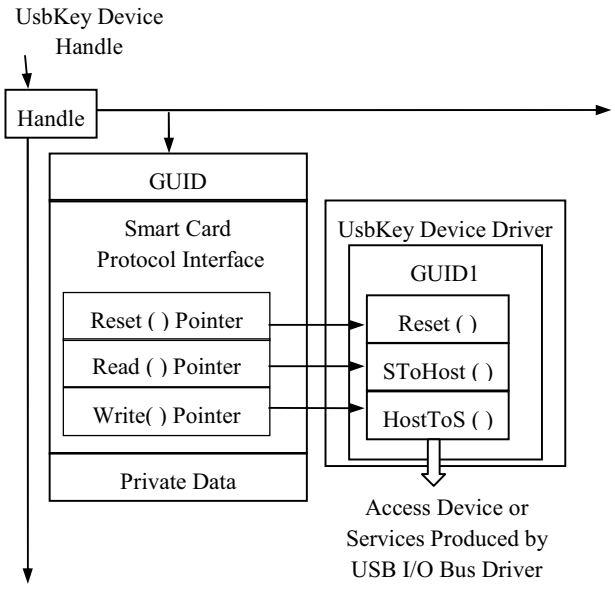

Figure 4. Construction of a Smart Card Protocol

The protocol is composed of a GUID and a protocol interface structure. The UsbKey device driver that produces a protocol interface will maintain additional private data fields. The Smart Card protocol interface structure itself simply contains pointers, such as function pointer Reset, Read and Write, to the protocol functions. The protocol functions are actually contained within the UsbKey device driver with the name Reset, SToHost and HostToS respectively. The UsbKey device driver produces EFI_SMARTCARD_IO_PROTOCOL protocol. The main purpose of function SToHost( ) and HostToS( ) is to build a information channel between host and smart card inside the UsbKey device. Where the main purpose of function HostToS( ) is to build information

channel from host to the smart card device, while function SToHost ( ) is verse visa. The stack similar to TCG software stack (TSS) [10] can be build based on these two functions in the UsbKey device driver. Then the function of user authentication, key storage, integrity check provided by UsbKey device can be used by application. Furthermore, trusted chain can be built with UsbKey as the hardware root of trust.

5. Conclusion

Over time, developers have had to continually extend the functionality of BIOS. However, BIOS presents significant problems for today's computers. EFI is an OS-and platform-independent boot and preboot interface. It lies between the OS and platform firmware, and addresses many of the limitations of BIOS. EFI provides developers with an alternative, flexible, maintainable, extensible, and platform- and OS independent boot method for embedded systems, desktops, workstations, and servers.

UsbKey device is a trusted device for the platform built on common PCs current in use. The UsbKey device is a useful replacement for the TPM defined by TCG. As compared with TPM defined by TCG, UsbKey has the main functions as TPM. It can function as root of trust for the platform. The driver for UsbKey must be developed before it can be used by platform application. This paper focuses on the design and implementation of driver for UsbKey based on EFI driver model. It is a basement for the TSS. A trusted chain can built based on this UsbKey device. Few researchers work on EFI since EFI is a new technology. This paper can provide useful reference for researchers in this field.

6. Acknowledgments

This material is based upon work supported in part by the 863 Foundation of the Ministry of Science and Technology of the People's Republic of China (2007AA012410 & 2007AA012177), Science and Technology Foundation of Beijing Municipal Commission of Education (KM200811232013) and Science and Technology Foundation of Beijing Jiaotong University (2008RC021).

7. References

[1] Trusted Computing Group, TCG PC Specific Implementation Specification, Version 1.1, Aug. 18th, 2003.

[2] Michael Kinney. "Solving BIOS Boot Issues with EFI". *Intel Developer UPDATE Magazine*. Sep.,2000.

[3] Intel Corporation. Extensible Firmware Interface Specification Version 1.10, Dec 1th, 2002

[4] Intel Corporation. Driver Writer's Guide for UEFI 2.0, Draft for review, Revision 0.95 Apr. 10th, 2007

[5] Cypress Semiconductor Corporation, CS5954AM USB Controller for NAND Flash Revised May 28, 2002

[6] IdeaBank™ Ltd. in JiangSu Province, ICOS /PK V1.0 Technical manual (Basic Command), Dec.2002

[7] Vincent Zimmer, Michael Rothman and Robert P.Hale, Beyond BIOS. Implementing the Unified Extensible Firmware Interface with Intel's Framework, Intel Press 2006.

[8] Intel Corporation. Universal Host Controller Interface (UHCI) Design Guide, Revision 1.1, Mar. 1996.

[9] Open Host Controller Interface Specification for USB, v. 1.0a, Sept. 1999, www.intel.com/technology/usb/ehcispec.htm.

[10] TCG Software Stack (TSS) Specification Version 1.2 Level 1, Part1: Commands and Structures, January 6, 2006.