Design and Implementation of Portable TPM Device Driver based on Extensible Firmware Interface

PENG Shuanghe and HAN Zhen
School of Computer and Information Technology, Beijing Jiaotong University
Beijing 100044,Beijing,China
Email: (shhpeng,zhan)@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. A portable TPM device is proposed and designed in our lab in this context. The portable TPM is a device which capabilities combined with the mass storage feature of USB stick and smart card. How to build the chain of trust using TPM based on legacy BIOS is a focus in the past several years. Extensible Firmware Interface (abbreviated as EFI) is intended as a significantly improved replacement of the old legacy BIOS. How to build the chain of trust using portable TPM based on EFI is what we focus on. Among which, the driver for the portable TPM device is a key part. It is a basement for the TPM Software Stack and secure application. This objective is to design and implement the driver of portable TPM based on EFI to provide root of trust for trusted platform.**

***Index Terms*—Portable TPM; Device Driver; Extensible Firmware Interface;**

I. 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. The TPM is designed to be tamper resistant but currently cannot protect against hardware based attacks, e.g. the monitoring of the bus that connects the TPM to the motherboard. TPM is embedded into the motherboard as defined by TCG. Currently PC is the common platform in the information system. 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 portable TPM is designed to solve this problem. The interface between the portable TPM 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 portable TPM can provide the same capabilities such as integrity measurement, cryptographic key services, protected storage and endorsement services like TPM defined by TCG.

BIOS is a key part of the software chain for PCs. Its drawbacks and limitations of openness and criteria make its poor security. Its specifications have not kept up with the evolution of the technology. With BIOS hitting the wall, Intel spearheaded the EFI (Extensible Firmware Interface) to solve these problems[2]. The EFI is a specification that defines a software interface between an operating system and platform firmware. EFI is intended as a significantly improved replacement of the old legacy BIOS firmware interface historically used by all IBM PC compatible personal computers.[3]

EFI contains an extensibility mechanism that will allow future media devices to be supported. Since we use portable TPM as a root of trust on current PC platform, if an EFI firmware is used instead of the BIOS, how to design and implement driver for portable TPM based on EFI is what this paper focuses on.

II. 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 [4], [5]. The entry point for a driver that follows the EFI Driver Model must follow some strict rules. It 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.

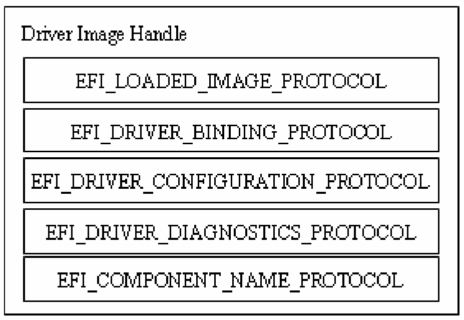
Driver Image Handle
EFI_LOADED_IMAGE_PROTOCOL
EFI_DRIVER_BINDING_PROTOCOL
EFI_DRIVER_CONFIGURATION_PROTOCOL
EFI_DRIVER_DIAGNOSTICS_PROTOCOL
EFI_COMPONENT_NAME_PROTOCOL

Fig. 1. Driver Image Handle

III. IMPLEMENTATION OF THE PORTABLE TPM

Since portable TPM 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.

USB mass storage device is widely used with the feature of portability, plug and play. While the security issues of the USB stick is headachy. When it is lost, the data can be accessed at will. The data and applications stored in the flash memory can't be protected.

In order to integrate the feature of both Smart Card and Mass Storage Disk, portable TPM is designed and implementation in our Lab. The portable TPM can provide both encryption component and protected storage space as defined by TCG with the feature of both Tamper Resistant Module and Mass Storage. It can provide user with multiple access rights, electronic signature and encryption key functions as well as a storage and backup system for sensitive data.

In the following section, hardware of portable TPM is first given and then the software part based on EFI is discussed in detail.

A. Hardware Components of portable TPM

Figure 2 shows the Functional Block Diagram of the portable TPM. The portable TPM is mainly made up of six components: USB Connector, USB Transceiver, SIE (Serial Interface Engineer), Processor, Smart Card and Flash Storage.

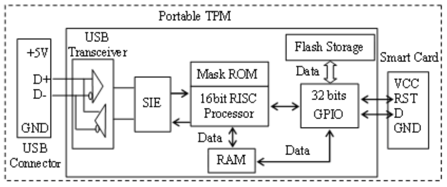

Fig. 2. Functional Block Diagram of the Portable TPM

Flash Storage chip and Smart card is two main components of the portable TPM device. We select a smart card [6] 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. USB connector is responsible for electrical connection between host and the portable TPM device. USB transceiver provides an information channel for command, data and status between host and the portable TPM. Serial Interface Engineer is a main part of USB controller. Its main task is to analyze USB protocol and transmit the data down to processor in the USB controller. Micro Processor is also part of USB controller. It can translate the data from host to the portable TPM device into different format according to the Operation Code field in the ATAPI [7] packet. When the value of Operation Code field is from 0x00 to 0xEF, then the data is from host to the flash memory chip, otherwise when the value of Operation Code field is from 0xF0 to 0xFF the data is from host to the smart card chip. When the data direction is from device to host, the translation process by the micro processor is verse visa.

B. Portable TPM Driver Stack

Figure 3 below shows the portable TPM driver stack and the protocols that the portable TPM 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). We selected UHCI in the prototype implementation.

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 portable TPM. 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. Because

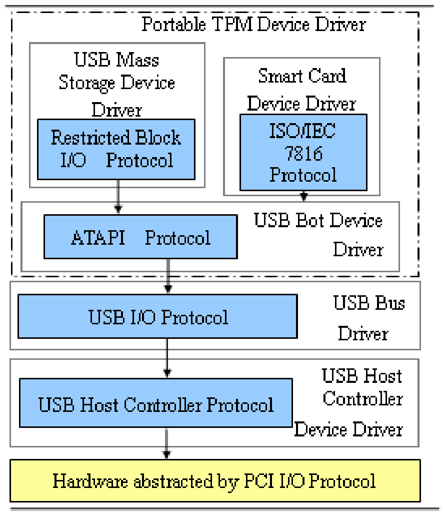
Portable TPM Device Driver
USB Mass Storage Device Driver
Restricted Block I/O Protocol
Smart Card Device Driver
ISO/IEC 7816 Protocol
ATAPI Protocol
USB Bot Device Driver
USB I/O Protocol
USB Bus Driver
USB Host Controller Protocol
USB Host Controller Device Driver
Hardware abstracted by PCI I/O Protocol

Fig. 3. Portable TPM Driver Stack

there are two types of USB command interfaces, we split the device driver in this example into two layers to support USB mass storage device. The first layer will consume the EFI_USB_IO_PROTOCOL and produce EFI_USB_ATAPI_PROTOCOL, which is a generalized interface. It depends on which command interface specification that the controller is based. Currently, the major two types are the USB Bulk-Only Transport (BOT)[9] driver and USB Control/Bulk/Interrupt Transport (CBI)[10]driver. We select BOT in the prototype implementation. The second layer is composed of two parts, one is the USB mass storage driver, the other is the Smart Card device driver. The USB mass storage driver will consume EFI_USB_ATAPI_PROTOCOL and produce a restricted EFI_RESTRICTED_BLOCK_IO_PROTOCOL. The USB Smart Card device driver will also consume EFI_USB_ATAPI_PROTOCOL and produce EFI_SMARTCARD_IO_PROTOCOL. The restricted EFI_RESTRICTED_BLOCK_IO_PROTOCOL is a restricted version of normal EFI_BLOCK_IO_PROTOCOL with ReadBlocks(), WriteBlocks() functions being called conditionally, i.e. access to data stored in the flash memory must be authenticated.

Since both mass storage device driver and smart card device driver consume ATAPI Protocol in the stack, the command packet format from host to these two modules is just the same. Then how to distinguish these two type commands is a problem. In the implementation, Bulk Only Transfer Command Block Wrapper packet is used. In order to distinguish these two modules, we extended the value range of the Operation Code field in the ATAPI Protocol command. The reserved value is utilized to achieve the goal.

When the Operation Code is from 0xF0 to 0xFF, then the command is from host to smart card, otherwise it is a common ATAPI Protocol command concerned with a mass storage device.

1) Functional Interface between Host and Portable TPM: Based on the portable TPM Driver Stack as Figure 3 shows, the functional interface between host and portable TPM is as Figure 4 shows.

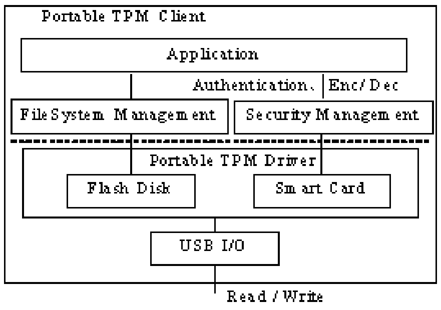

Fig. 4. Functional Interface between Application and Portable TPM

The client software for portable TPM is composed of two parts, one is file system management and the other is security management. The security module is responsible for secure function such as authentication, keys negotiation and mathematical computations regarding to trusted service as defined in TPM Software Stack (TSS) by TCG. The file system module is a controlled file system. User must be authenticated before data is accessed in the flash chip. The interaction between these two parts can provide integrity measurement, cryptographic key services, protected storage and endorsement services just as TPM can do.

2) Implementation of Portable TPM Driver based on EFI: According to EFI Driver Model, portable TPM device Driver based on EFI is built as follows: First the portable TPM device Driver needs to produce a Driver Binding Protocol. In the start() service of the Driver Binding Protocol, EFI_SMARTCARD_PROTOCOL and EFI_RESTRICTED_BLOCK_IO_PROTOCOL is installed in the handle. Figure 5 below shows the portable TPM device handle and protocols from the handle database that is produced by the portable TPM device driver.

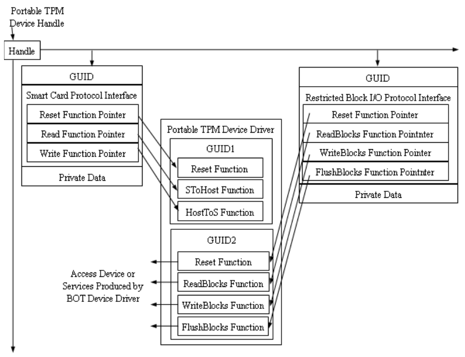
Portable TPM Device Handle
Handle
GUID
Smart Card Protocol Interface
Reset Function Pointer
Read Function Pointer
Write Function Pointer
Private Data
Portable TPM Device Driver
GUID1
Reset Function
SToHost Function
HostToS Function
GUID2
Reset Function
ReadBlocks Function
WriteBlocks Function
FlushBlocks Function
Access Device or Services Produced by BOT Device Driver
GUID
Restricted Block I/O Protocol Interface
Reset Function Pointer
ReadBlocks Function Pointnter
WriteBlocks Function Pointer
FlushBlocks Function Pointnter
Private Data

Fig. 5. Construction of a Smart Card protocol and restricted Block I/O Protocol

The protocol is composed of a GUID and a protocol interface structure. The portable TPM 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 portable TPM device driver with the name Reset, SToHost and HostToS respectively. The portable TPM device driver produces EFI_SMARTCARD_PROTOCOL and EFI_RESTRICTED_BLOCK_IO_PROTOCOL protocol. The main purpose of function SToHost() and HostToS() is to build a information channel between host and smart card inside the portable TPM 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 TSS can be build based on these two functions in the portable TPM driver.

C. Portable TPM Driver Connection Process

All UEFI drivers that adhere to the UEFI Driver Model follow the same basic procedure when the driver is loaded. When the portable TPM device is inserted into the USB interface, the portable TPM driver connection process is just as Figure 6 shows:

The handle status change as follows:

First, when the EFI system called LoadImage(), driver file image is loaded into the system. It will install a Driver Binding

Protocol on the image handle from which it was loaded. It may also update a pointer to the EFI_LOADED_IMAGE Protocol's Unload() service and install Component Name Protocol so its name is visible to any operator. The driver will then exit from the entry point, leaving the code resident in system memory. The Driver Binding Protocol provides a version number and the following three services: Supported(), Start() and Stop(). These services are available on the portable TPM device driver's image handle after the entry point is exited. Later on when the system is ”connecting” drivers to the portable TPM device in the system, the driver's Driver Binding Protocol Supported() service is called. The Supported() service is passed a controller handle. Quickly, the Supported() function will examine the controller handle to see if it represents a device that the driver knows how to manage. In the portable TPM context, it will test to see if this driver supports ControllerHandle that contains a USB I/O protocol. And then it will use the USB I/O protocol interface to see the Controller is the UsbKey controller that can be managed by this driver. If so, it will return EFI_SUCCESS. The system will then start the driver by calling the driver's Start() service, passing in the supported controller handle. In the Start() service, EFI_SMARTCARD_PROTOCOL and the restricted protocol EFI_RESTRICTED_BLOCK_IO_PROTOCOL is installed onto the controller handle. The driver can later be disconnected from a controller handle by calling the Stop() service as showed by dashed line in Figure 6.

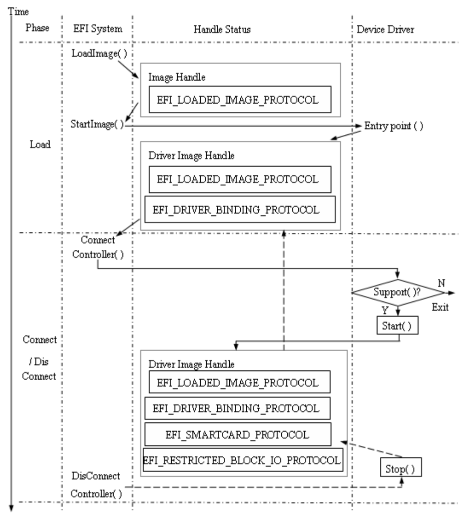

Fig. 6. Portable TPM Driver Connection Process

IV. Conclusion

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. Portable TPM is a trusted device for the platform built on common PCs current in use. The portable TPM device is a useful replacement for the TPM defined by TCG. This paper focuses on the design and implementation of driver for portable TPM based on EFI driver model. It is a basement for the TSS. A trusted chain can built based on this portable TPM. Few researchers work on EFI since EFI is a new technology. This paper can provide useful reference for researchers in this field. In the coming future, trusted recovery is the focus of our work. The mechanism of trusted recovery can assure the recovery capability of the system when failures occur.

Acknowledgment

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 (2007AA01Z410 & 2007AA01Z177), 973 Research Foundation (2007CB307101), Science and Technology Foundation of Beijing Municipal Commission of Education (KM200811232013) and Science and Technology Foundation of Beijing Jiaotong University (2008RC021).

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] Vincent Zimmer, Michael Rothman and Robert P.Hale, Beyond BIOS. Implementing the Unified Extensible Firmware Interface with Intel's Framework, Intel Press 2006.
[6] IdeaBankTM Ltd. in JiangSu Province, ICOS /PK V1.0 Technical manual (Basic Command), Dec.2002.
[7] USB Implementers Forum. Universal Serial Bus Mass Storage Class UFI Command Specification, Revision 1.0, Dec. 14 th, 1998.
[8] Intel Corporation. Universal Host Controller Interface (UHCI) Design Guide, Revision 1.1, Mar. 1996.
[9] USB Implementers Forum. Universal Serial Bus Mass Storage Class Bulk-Only Transport, Revision 1.0, Sep. 31th, 1999.
[10] USB Implementers Forum. USB Mass Storage Class Control/Bulk/Interrupt (CBI) Transport, Revision 1.1, Jun. 23th, 2003.