Analyzing trusted platform communication *

Klaus Kursawe, Dries Schellekens **, and Bart Preneel

Katholieke Universiteit Leuven
Department Electrical Engineering-ESAT/SCD-COSIC,
Kasteelpark Arenberg 10, 3001 Heverlee, Belgium
{klaus.kursawe,dries.schellekens,bart.preneel}@esat.kuleuven.be
http://www.esat.kuleuven.be/cosic/

Abstract. In this paper we discuss the analysis of trusted platform communication. While the trusted platform module itself is considered reasonably tamper resistant, the communication channel between this module and the rest of the trusted platform turns out to be comparatively insecure. Passive attacks can be mounted on the communication interface with fairly inexpensive equipment and allow eavesdropping of critical information. Performing active manipulation on the communication bus could provide an even stronger attack scenario, resulting in a circumvention of the whole chain of trust provided by trusted platforms. At this stage, our research has been limited to passive attacks.

1 Introduction

While many security problems can be resolved by hardened operating systems and applications, some issues require hardware means – it is, for example, impossible to implement a trusted boot process in pure software [2, 4]. Also, as some applications depend more and more on other (remote) platforms, trust relationship between several platforms need to be established. This too is difficult to achieve in software.

To address these problems, the Trusted Computing Group (TCG) has developed an industry standard for security hardware usable in a wide range of platforms and applications. This standard is supposed to serve as a relatively cheap *root of trust* that the operating system and higher level applications can build upon, both for hardening the platform and for establishing remote trust.

To this end, a low performance cryptographic coprocessor, the *Trusted Platform Module* (TPM), is added to the corresponding platform. Besides a number of standard cryptographic operations, the TPM can also be used to make provable statements about the host platform, such as the properties of certain keys or the boot sequence.

In some applications, such as Grid Computing or Digital Rights Enforcement, this implies that the platform owner has an interest in lying about some properties. Thus, we face an attacker that may have access to the hardware itself rather than being restricted to a pure software attack.

While the TPM itself can be expected to be reasonably secure against standard attacks, the communication channel between TPM and CPU is mainly unsecured. The TPM is usually connected using the *Low Pin Count* (LPC) bus, which can be monitored using relatively inexpensive equipment.

As the TPM learns all about its environment through this channel, and communicates critical information (it can, for example, not perform any bulk encryption by itself and thus needs to transfer keys to the CPU), this gives an interesting point to attack a trusted computing architecture.

In this paper, we investigate the possibilities to attack the communication channel between the TPM and the CPU. For the time being, we restrict ourselves to *passive* attacks,

* This work was supported by the Concerted Research Action (GOA) Mefisto-2000/04 of the Flemish Government.
** Research funded by a research grant of the K.U.Leuven.

i.e., we only monitor the communication. This way, we can get access to some key material, and analyze an application's precise usage of the TPM.

The next step would be to perform *active* attacks, i.e., modify the communication to fool the TPM about the environment it is operating in. Such an attack may be able to completely circumvent the remote trust concept, as the TPM itself has no reliable information on the platform anymore, and may convince a TPM to reveal keys it should normally keep concealed.

Related work Physical probing attacks are common to smart cards and other tamper resistant devices; e.g., to probe memories and data busses inside a smart card [1, 5]. The attacks we describe are far easier to perform, as no special equipment is needed and external access to a TPM is sufficient, and still can have big implications. Of course, by de-packaging a TPM and probing it internally, the *Endorsement Key*, which in the security model of the TCG should never leave the TPM, could be extracted.

Others have analyzed trusted platform communication, but not as a means of attack. In order to write a Linux device driver for the Infineon TPM, people of the Reliable Software Group at UCSB have used a logical analyzer on the LPC bus to adapt the existing Atmel driver [15].

Organization of this paper The outline of this paper is as follows. In Section 2, we start with a description of our experiment setup for passive attacks. In Section 3, we discuss all aspects of the analysis of trusted platform communication. We propose some countermeasures against passive attacks in Section 4. Section 5 lists our plan for future work, highlighting some ideas regarding active attacks. Ultimately, we conclude in Section 6.

2 Experiment setup for passive attacks

At the moment various desktop computers and laptops are available on the market with a TPM; for an extensive list of TPM manufacturers and implementations we refer to [9]. The major vendors are HP/Compaq, IBM and Intel – all founding members of the Trusted Computing Platform Alliance (TCPA), the predecessor of the TCG – as well as Dell. HP ships its machines with an Infineon TPM, while Intel sells a number of motherboards that optionally are available with this same module. IBM on the other hand used an Atmel module in older models, and currently uses a National Semiconductor TPM integrated into a Super I/O chip. Finally, Dell utilizes a Broadcom gigabit ethernet controller with integrated TPM.

For our experiment we used an IBM ThinkCentre M50, mainly for two reasons. First, the Atmel TPM in it is installed on a *daughter board*, so not on the mainboard or integrated into another chip. Therefore its communication interface is easily accessible. Furthermore, IBM machines have good Linux support; the TPM driver written by IBM [6] is included in the latest mainstream kernel, and IBM's Linux Technology Center is also developing an open source TCG Software Stack (TSS), called TrouSerS [7].

An Agilent 16900 Logic Analysis System was used to analyze the communication. As the daughter board is mounted on the motherboard using a connector and the pins of this connector are relatively far apart, it is very simple to put probes on the different signals.

In a first phase we used a modified version of the Linux TPM device driver to understand the transfer protocols involved and identify the meaning of the different signals on the communication bus; e.g., to determine the assignment of the 4 data bits on the bus. In this way we were able to log all commands the TCG Software Stack sends to the TPM, and the responses it gets back.

In a second step we observed the trusted platform communication during startup. However, during the boot process too much traffic is generated on the communication bus to store all of it, as the BIOS firmware is loaded over the same bus. The logic analyzer can trigger on the first packet to the TPM, and fill its memory from that moment in time, but there is no feature to only log TPM packets – so selectively store datagrams. It is possible though, but not really practical, to startup the PC many times, and use a different trigger

condition to ultimately capture all LPC traffic during startup; post processing of this data is required to filter out the TPM communication. We considered two options to efficiently address the problem:

- Implement a dedicated logging device that only logs communication addressed to the TPM. Such device has to analyze the packets on the bus, and only store the ones from and to the TPM. The logged data – latter or simultaneously – has to be sent to a computer for further analysis.
- Add a filtering device in front of the logic analyzer (i.e., pre processing). This device will only forward packets to the logic analyzer if they satisfy certain criteria, namely the TPM as destination.

Boths solution can be implemented in software – on a microcontroller – or in hardware – on a FPGA. We choose the second solution because of its simplicity, and implemented this functionality in HDL on a Xilinx Virtex-II Pro.

3 Analysis of trusted platform communication

The trusted platform communication can be analyzed at – what can roughly be called – three levels (or layers in network stack terminology).

Application level. The TCG has standardized the communication with the trusted platform module: commands, responses, packet format, authentication protocols, etc. The analysis at this level is generic and not limited to a certain experimental setup.

Transport level. How the packets are transferred is platform and manufacturer dependent. Most TPM manufacturers use the Low Pin Count bus interface to do so, and therefore are only suited for the PC client platform. Atmel additionally provides a System Management Bus (SMBus) interface on its older TPMs, whereas the newer modules no longer do so. Atmel's module for embedded devices only uses this second interface, because such devices typically lack a LPC bus. Therefore the analysis of this layer depends upon the TPM manufacturer, but is rather similar for all LPC bus based implementations.

Physical level. It is up to the PC manufacturer how to actually install a TPM in its machines. Some mount it on the motherboard, or use one that is integrated into an other peripheral, others want it to be a build option and use a daughter board. Here, the analysis is focussed on a specific TPM installation.

In the next sections we provide an analysis at these different levels, starting from the lowest level. The installation of the Atmel TPM in our experiment PC is used as an example.

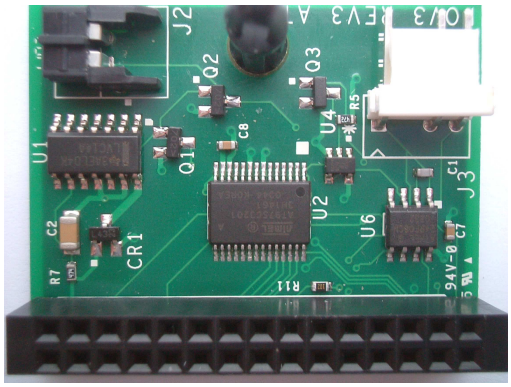

Fig. 1. TPM daughter board in IBM ThinkCentre M50. The chip U2, in the middle of the board, is a Atmel TPM. Right of it, IC U6 is an Asset Identification EEPROM. All communication to the motherboard happens over the 30 pin connector at the bottom.

Table 1. Interconnection of Atmel TPM and daughter board connector. To avoid any confusion, LAD3 is connected to the connector pin 9, LAD2 to pin 22 and so forth. Some signals are connected to the ground level because their functionality is unused; of course GND originates from connector pin 1.

TPM Pin Name	Function	Connector Pin
GND	Ground	1
LAD[3:0]	LPC Command, Address and Data	9,22,8,23
LFRAME#	LPC Frame Indicator	28
LCLK	33 MHz PCI Clock	29
LRESET#	System Reset (active low)	5
SIRQ	Serialized IRQ	11
VBB	Battery Input	16
VCC	3.3V Supply Voltage	18
CLKRUN#	Clock Control	GND
IOA	Input/Output A (SMBCLK)	GND
IOB	Input/Output B (SMBDAT)	unknown
IOC	Input/Output C	GND
Xtamper	External Tamper Detect	GND
Xtal1	32.768 kHz Crystal	GND
Xtal2	32.768 kHz Crystal	unknown

3.1 Trusted platform daughter board

Some reserve engineering effort is needed to determine the electrical specification and function of all signals on the daughter board. The daughter board inside the IBM ThinkCentre M50 is a simple circuit board containing 2 chips (see Figure 1):

- Atmel AT97SC3201: Trusted Platform Module, TCG/TCPA version 1.1b compatible.
- Atmel AT24RF08C: Asset Identification EEPROM with dual access (wired serial port and wireless RFID port). While placed on the same daughter board, this is in fact unrelated to the trusted computing architecture.

In Table 1 we describe how all pins of the Atmel TPM are wired to the connector of the daughter board. We start numbering the connector pins (as viewed on Figure 1) from the left lower corner (this is pin 1), and continue counter clockwise (thus the upper left pin is 30). The interconnection on the PCB was determined using a multimeter.

Some of the features of the Atmel TPM are unused: power management (the bus clock can normally be stopped using CLKRUN#), SMBus interface, and external tamper detection. The TPM will still detect if it has been removed from the PC by monitoring the motherboard battery (VBB). However, we suspect that it is possible to temporally remove the daughter board by connecting it to an external battery.

For the pin configuration of the Atmel chip we refer to the latest version of the data sheet [3]; the previous version, which was available when we did most of the reverse engineering work, did not contain this information. The complete documentation is still only available under a non-disclosure agreement.

3.2 Low Pin Count bus

From the analysis at the physical analysis it is clear that the LPC bus, and not the SMBus, is used in our experiment machine to transfer TCG packets. This is the case in most other implementations – especially as other TPM manufacturers only support the LPC bus. The use of the LPC bus will even be made an obligation by the TCG and is being standardized in the TCG TPM Interface specification.

The Low Pin Count interface specification [8] was developed by Intel for ISA-less PCs. It offers a cost-effective and easy bus, with only 7 required (LAD[3:0], LFRAME#, LRESET# and LCLK) and some optional signals (e.g., CLKRUN#). The bus use the PCI 33 MHz clock, and allows various transfer protocols (memory, I/O, DMA, firmware memory and bus master).

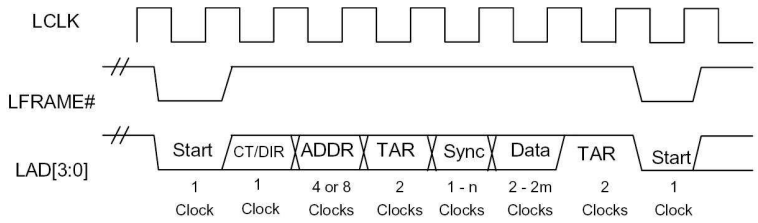

Fig. 2. Typical LPC bus timing. CT/DIR indicates cycle type and direction, and TAR denotes turn-around.

Data transfers on the LPC bus are serialized over a 4 bit bus. `LFRAME#` is used by the host to start or stop transfers, and by peripherals to determine when to monitor the bus for a cycle. The `LAD[3:0]` bus communicates address, control, and data information serially. The general flow of cycles is visualized in Figure 2 and goes as follows:

1. A cycle is started by the host when it drives `LFRAME#` active, and puts appropriate information on the `LAD[3:0]` signal lines.
2. The host drives information relative to the cycle, such a cycle type (memory, I/O, DMA), read/write direction, and address.
3. The host optionally drives data, and turns the bus around to monitor the peripheral for completion of the cycle.
4. The peripheral indicates the completion of the cycle by driving appropriate values on the `LAD[3:0]` signal lines, and potentially drives data.
5. The peripheral turns the bus around to the host, ending the cycle.

The use of the LPC bus for trusted platform communication is currently not standardized. In case of the Atmel TPM this is not mentioned in the publicly available data sheet either, yet the Linux device driver (in combination with the logic analyzer) reveals that *port mapped I/O* is used: I/O write cycles are used to send requests to the TPM, and I/O read to get back the responses. The forthcoming TCG TPM Interface specification, on the other hand, is said to standardize *memory-mapped I/O* over the LPC bus.

Table 2 gives the definition of all I/O cycle fields. The exact sequence of the fields for an I/O read cycle matches the generic timing on Figure 2. For I/O cycles the address field is 16 bits (so 4 clock cycles), and the data returned is 1 byte (thus 2 clocks wide). A write cycle is very similar: after the address is transferred, a data byte is sent, and the bus is handed over to the TPM, which will add wait states until it is ready to turn around the bus.

The two I/O addresses in Table 2 are specific to the Atmel TPM, and were found in the Linux device driver. The procedure to send and receive TCG datagrams over the LPC bus is documented as well in this open source device driver. The chip does not support interrupts, so the status of the device must be determined by *polling*. The transfer process more or less goes as follows:

1. A TCG request is transferred using multiple I/O Writes to the TPM *data port*.
2. The device driver will repeatedly check the status (using I/O Reads of the *status port*) to see if the TPM is still busy and when data is available.
3. Once data is available, the TCG response can be read byte by byte (using I/O Reads from the data port). As the driver does not know the size of the response, it will always check the status of the TPM to determine data is still available, before reading the next byte.
4. The transfer has completed whenever no data is left.

3.3 TCG communication

At the command level, trusted platform communication is strictly standardized by the Trusted Computing Group in various specifications. The Trusted Platform Module Main

Table 2. LPC I/O cycle field definitions. The value of `LAD[3:0]` is given in hexadecimal format, and the 16 bit I/O address (represented with 4-digit hex code) is transferred with the most significant nibble first. During the turn-around time `LAD[3:0]` will be driven to `1111` during the first clock, and tri-state during the second clock. The TPM needs to assert wait states, and does so by driving `0110` (long SYNC) on `LAD[3:0]` until it is ready (so $n-1$ clock cycles); when ready, it drives `0000` (1 clock).

Field	# Clocks	Description	`LAD[3:0]`
START	1	*Start of Cycle*	`0x0`
CT/DIR	1	*Cycle Type/Direction*	
		I/O Read	`0x0`
		I/O Write	`0x2`
ADDR	4	*Address*	
		TPM data port (Atmel specific)	`0xE000`
		TPM status port (Atmel specific)	`0xE001`
TAR	2	*Turn-Around*	`0xFF`
SYNC	n	*Synchronize*	
		Ready	`0x0`
		Long Wait	`0x6`
DATA	2	*Data byte*	
		least significant nibble first	`Data[3:0]`
		most significant nibble next	`Data[7:4]`

specification [12] describes the functionality of the TPM: design principles, cryptographic algorithms and protocols, data structures, commands, etc. In the latest version [14], the main specification has been broken up in three separate parts again, without significantly changing the structure.

The PC Specific Implementation specification [13] documents how a trusted platform module is integrated into the Personal Computer platform, and mainly focusses on the requirements for the BIOS, which forms the *Core Root of Trust for Measurement* (CRTM) in case of the PC platform. The TCG is working on an updated version of this document for the conventional BIOS, but is also developing a standard for EFI (Extensible Firmware Interface) based platforms.

For our analysis of the LPC interface, we used a single, fixed command, that queries the trusted platform module for certain information (e.g., manufacturer, version, algorithm support): TPM_GetCapability. This command is available even for deactivated TPMs and thus is a natural choice for a first analysis.

In a second phase, we monitored the – not rigidly specified – communication during startup. First the BIOS checks if the TPM is ready by querying the status port (this is Atmel specific). As is to be expected, it next performs multiple TPM_PcrExtend commands. This instruction stores a new hash value of some component of the boot process to a named *Platform Configuration Register* (PCR) and consequently forms the basis of the integrity collection process – one of the main features of trusted computing. Later these integrity metrics can be reported to external parties, in order that they get convinced the platform is running in an well defined and approved configuration.

In Section 5 we describe some of the future work we plan to do at this level. In particular, we want to try gaining access to key material and analyze how closed source applications precisely use the TPM.

4 Countermeasures

There are a number of countermeasures available against attacks such as ours; some of them have partially been implemented already, though related attacks may not be eliminated in the foreseeable future.

The latest specification of the TCG (version 1.2) [14] already allows the TPM to build an encrypted channel to other devices. It is, however, unlikely that the processor itself will

be able to securely communicate with the TPM. Also, while sufficient to protect keys, this channel does not prevent active attacks, such as fooling the TPM into believing there was a platform reset.

As already done in newer IBM machines, the TPM can be *integrated* into other chipsets, such as the Super I/O chip. This essentially eliminates access to the LPC bus – the BIOS firmware should of course be embedded into the chipset as well, otherwise the LPC bus can still be monitored at the BIOS flash memory chip. An attack similar to ours would thus require measuring and manipulating the PCI bus which, while possible, involves significantly more effort and cost. It is, however, unclear what practical implications such an integration has. External validation and certification of a TPM will be significantly more difficult, and hardening the TPM against chip level attacks may be more difficult. As for encrypted communication, this may not solve all problems either. For example, it still may be possible to reset the entire Super I/O chip without resetting the platform, though we have not done this in practice yet.

Finally, and most importantly, protocol designers should be aware of the limitations of a trusted computing architecture. While it is possible to build highly secure TPMs, and the TCG concept explicitly allows an application to distinguish between different TPM implementations, the vast majority of TPMs will offer only a limited protection against hardware based attacks. As with many security techniques, it is important to be aware of the protection the technique offers, and not use it for purposes it was never designed for.

5 Future work

It should be rather straightforward to do the same analysis on other TPM implementations, on the condition that the LPC bus is used to transfer TCG packets. More interesting would be to look at integrated TPM solutions, namely National Semiconductor's Super I/O chip and the Broadcom BCM5752 gigabit ethernet controller with integrated TPM. Do these provide adequate protection against the described passive attacks? Is there no way to monitor the communication externally?

For now, our analysis is limited to a controlled environment – sending known commands in Linux to the TPM and observing them on the communication channel – and to the boot process. In a next step, we want to really extract key material from applications making use of trusted computing functionality. Some candidate software includes IBM Client Security Software and HP ProtectTools; both are Windows implementations of a TCG Software Stack, and, for instance, can store Encrypted File System keys in a TPM. Future operating systems will be designed to use TCG features, so trusted platform functionality will no longer be a third party add-on. Microsoft Longhorn, the next generation of the Windows operating system, will, for instance, offer secure startup [10] and key storage [11]. There is some indication that Apple could use a TPM to limit the installation of their operating system to Intel based Macs alone, when they make the transition from the PowerPC architecture to Intel IA32 for their CPU.

Finally, actively manipulating the communication between CPU and TPM, rather than just observing the bus, allows for much stronger attacks. The main target we see is to fool the TPM about the platform status. The primary goal here is to manipulate the Platform Configuration Registers, in which the TPM stores hash values of the boot sequence. This violates one of the main security goals, namely to make provable statements about the platform configuration. There are several ways to access these values:

- If the TPM can be convinced that the platform is rebooting (i.e., it receives a reset signal), it will allow to overwrite the PC Registers with the new "boot sequence".
- If the TPM can be (partially) cut of during boot, it will never enter the original boot sequence into the PC Registers, which then again can be manipulated by a rogue program.
- Finally, it may be possible to manipulate the values entered during the original boot sequence, leading to a wrong content of the PC Registers.

In addition, the latest version of the TPM specification allows privileged processes to obtain more rights than normal processes. The identification of such privileged processes (for example, a microkernel running in a special processor ring) will be done by different addressing modes on the LPC bus. If these can be copied, any command can be given special privileges.

6 Conclusion

In this work, we analyzed to which extent a concrete implementation of TGC's Trusted Platform Module does satisfy the abstract requirements an application would expect such a module to have. Concretely, rather than considering an attack on the TPM itself, we examine the less protected communication channel between the TPM and its host.

We restricted ourselves to passive attacks, i.e., only measured the data sent over the bus. In a real attack scenario, this would mainly be useful to analyze the way programs use the TPM and to eavesdrop on keys used for bulk encryption. Additionally, the potential for an active attack can clearly be seen, and first experiments showed that the TPM does not seem to detect active manipulation on the LPC bus. Such attacks can go much further, for example by circumventing the TPM's integrity reporting mechanism.

It is therefore important to not trust the TPM as it is, but – depending on the application – also consider the concrete implementation (as some TPMs do appear to be stronger protected than others), and to be aware of the limits of the trusted computing concepts during application design.

Acknowledgements

The authors would wish to thank Kazuo Sakiyama for the huge help with the measurement setup and the FPGA implementation.

References

1. Ross Anderson, Markus Kuhn, *Low Cost Attacks on Tamper-Resistant Devices*, Security Protocol Workshop 1997, LNCS 1361, Springer-Verlag, pp. 125–136, 1997.
2. William A. Arbaugh, David J. Farber, Jonathan M. Smith, *A Secure and Reliable Bootstrap Architecture*, IEEE Symposium on Security and Privacy 1997, IEEE Computer Society, pp. 65–71, 1997.
3. Atmel, *AT97SC3201 Summary*, available at `http://www.atmel.com/products/Embedded/`
4. Michael Groß, *Vertrauenswürdiges Booten als Grundlage authentischer Basissysteme*, Verläßliche Informationssysteme, GI-Fachtagung, Informatik-Fachberichte 271, Springer-Verlag, pp. 190–207, 1991.
5. Helena Handschuh, Pascal Paillier, Jacques Stern, *Probing Attacks on Tamper-Resistant Devices*, CHES 1999, LNCS 1717, Springer-Verlag, pp. 303–315, 1999.
6. IBM Linux Technology Center, *Linux TPM Device Driver*, available at `http://tpmdd.sf.net`
7. IBM Linux Technology Center, *TrouSerS – An open-source TCG Software Stack implementation*, available at `http://trousers.sf.net`
8. Intel, *Low Pin Count Interface Specification*, available at `http://www.intel.com/design/chipsets/industry/lpc.htm`
9. Tony McFadden, *TPM Matrix*, `http://www.tonymcfadden.net/tpmvendors.htm`
10. Microsoft, *Secure Startup – Full Volume Encryption: Technical Overview*, available at `http://www.microsoft.com/whdc/system/platform/pcdesign/secure-start_tech.mspx`
11. Microsoft, *Trusted Platform Module Services in Windows Longhorn*, available at `http://www.microsoft.com/whdc/system/platform/pcdesign/TPM_secure.mspx`
12. Trusted Computing Group, *Main Specification Version 1.1b*, available at `https://www.trustedcomputinggroup.org/downloads/specifications/tpm`
13. Trusted Computing Group, *PC Specific Implementation Specification Version 1.1*, available at `https://www.trustedcomputinggroup.org/downloads/specifications/pcclient`
14. Trusted Computing Group, *TPM Specification Version 1.2*, available at `https://www.trustedcomputinggroup.org/downloads/specifications/tpm`
15. Fredrik Valeur, personal conversation, 2004.