Software-based TPM Emulator for Linux

Semester Thesis

Mario Strasser

Department of Computer Science
Swiss Federal Institute of Technology Zurich

Summer Semester 2004

Outline

1. Motivation and Goals
2. Trusted Computing
3. Trusted Platform Module
4. TPM Emulator
5. Conclusion

Outline

1. Motivation and Goals
2. Trusted Computing
3. Trusted Platform Module
4. TPM Emulator
5. Conclusion

Outline

1. Motivation and Goals
2. Trusted Computing
3. Trusted Platform Module
4. TPM Emulator
5. Conclusion

Outline

1. Motivation and Goals
2. Trusted Computing
3. Trusted Platform Module
4. TPM Emulator
5. Conclusion

Outline

1. Motivation and Goals
2. Trusted Computing
3. Trusted Platform Module
4. TPM Emulator
5. Conclusion

Status Quo

Status Quo

- Many controversies about TC, TCG/TCPA, TPM (*Fritz-Chip*), and DRM.
- IBM provides a TPM device driver for their ThinkPads as well as some example applications.
- Dartmouth College works on a TPM-based file integrity measurement module (*Enforcer*) for Linux.

Motivation, and Goals

"What I cannot create I do not understand." (R. Feynman)

Motivation

- Give people the means to easily explore TPMs for educational and experimental purposes.

Goals

- Implementation of a software-based TPM emulator for Linux in cooperation with IBM (David Safford, Jeff Kravitz) and Dartmouth College (Omen Wild).

Motivation, and Goals

"What I cannot create I do not understand." (R. Feynman)

Motivation

- Give people the means to easily explore TPMs for educational and experimental purposes.

Goals

- Implementation of a software-based TPM emulator for Linux in cooperation with IBM (David Safford, Jeff Kravitz) and Dartmouth College (Omen Wild).

Trusted Computing

- A Trusted (Computing) Platform is a platform that is trusted by local and remote users.
- A relationship of trust must be established between the user and the computing platform so that the user believes that an expected boot process, a selected operating system, and a set of selected security functions in the computing platform have been properly installed and operate correctly.

Overview

The Trusted Platform Module (TPM) is a hardware component that provides four major functions:

1. Cryptographic functions: RSA, (P)RNG, SHA-1, HMAC
2. Secure storage and reporting of hash values representing a specific platform configuration
3. Key storage and data sealing
4. Initialization and management functions (opt-in)

Auxiliary functions since version 1.2:

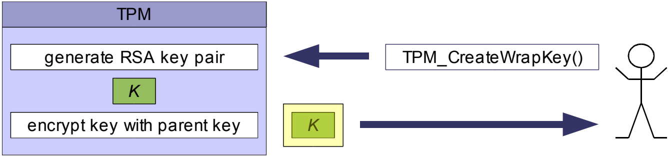
- Monotonic counters and timing-ticks
- Non-volatile storage
- Auditing

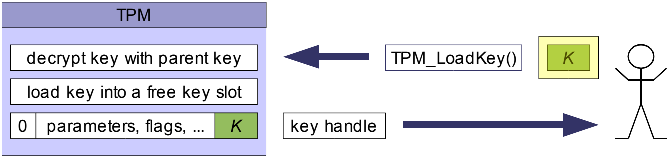
Platform Configuration

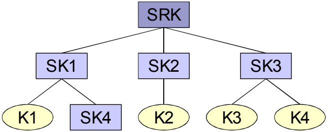
- Obtaining metrics of platform characteristics that affect the integrity of a platform and storing them into the PCRs.
- A PCR update includes history:
  PCR[n] $\leftarrow$ SHA-1(PCR[n] $+$ measured data).
- Transitive trust (inductive trust) is applied to extend the trust boundary during bootup:
  TPM, CRTM $\rightarrow$ BIOS $\rightarrow$ MBR $\rightarrow$ OS $\rightarrow$ Application.

PCR Index	PCR Usage
0	CRTM, BIOS, and Platform Extensions
1	Platform Configuration
2	Option ROM Code
3	Option ROM Configuration and Data
4	IPL Code (usually the MBR)
5	IPL Code Configuration and Data

Key Generation and Storage







Signing and Sealing

Binding Encryption of a message using a public key. The (migratable) private key is managed (stored) by the TPM.

Sealing Encryption of a message using a public key with additional binding to a set of platform metrics (system must be in a specific configuration to successfully unseal the data).

Signing Signing of a message digest using a signing only private key.

Sealed-Signing Signing of a message digest using a signing-only private key with inclusion of the current platform metrics.

Signing and Sealing

Binding Encryption of a message using a public key. The (migratable) private key is managed (stored) by the TPM.

Sealing Encryption of a message using a public key with additional binding to a set of platform metrics (system must be in a specific configuration to successfully unseal the data).

Signing Signing of a message digest using a signing only private key.

Sealed-Signing Signing of a message digest using a signing-only private key with inclusion of the current platform metrics.

Signing and Sealing

Binding Encryption of a message using a public key. The (migratable) private key is managed (stored) by the TPM.

Sealing Encryption of a message using a public key with additional binding to a set of platform metrics (system must be in a specific configuration to successfully unseal the data).

Signing Signing of a message digest using a signing only private key.

Sealed-Signing Signing of a message digest using a signing-only private key with inclusion of the current platform metrics.

Signing and Sealing

Binding Encryption of a message using a public key. The (migratable) private key is managed (stored) by the TPM.

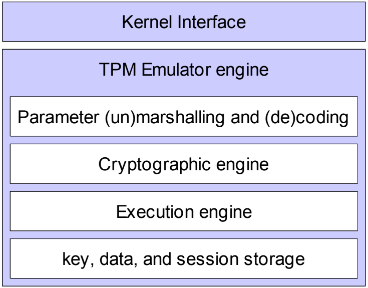
Sealing Encryption of a message using a public key with additional binding to a set of platform metrics (system must be in a specific configuration to successfully unseal the data).

Signing Signing of a message digest using a signing only private key.

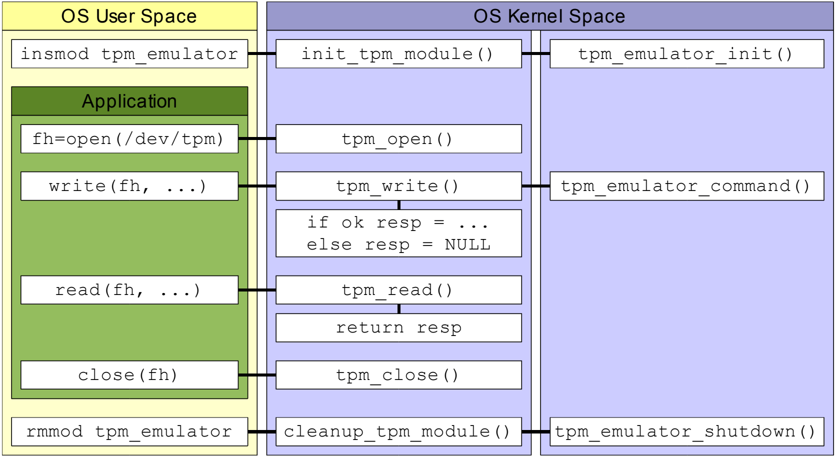
Sealed-Signing Signing of a message digest using a signing-only private key with inclusion of the current platform metrics.

Overview

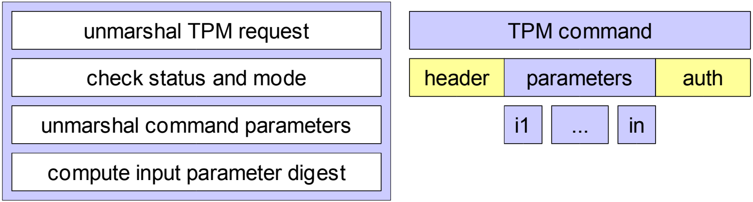
- Emulator is provided as a Linux kernel module (i.e., device driver for the device file `/dev/tpm`)
- One goal was to be compatible with the driver from IBM
- Two main parts: kernel interface and emulator engine
- Startup mode is specified by a module parameter
- Command serialization by means of semaphores



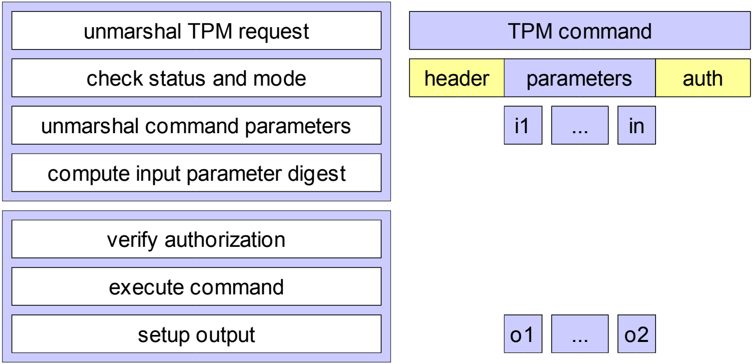
Command Execution I



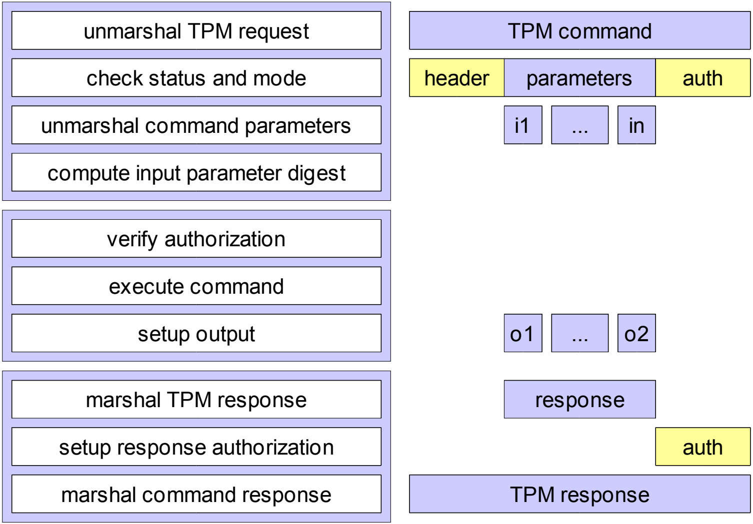
Command Execution II



Command Execution II



Command Execution II



Two Selected Problems I

Problem 1: Persistent Storage

- Seems to be easy, just write all data to hard disk.
- But file system is under the user's control – what if disk is not yet mounted or temporarily unmounted?
- As a general design rule: kernel functions should not directly write to hard disks!
- Possible solution: use user tool to (re)store TPM's data. Drawback: TPM knows when saving is advisable, not the user.

Two Selected Problems II

Problem 2: (Un-)Marshalling and command decoding

- Only five basic TPM types: BYTE, BOOL, UINT16, UINT32, UINT64, and BLOB (BYTE*).
- All non-basic TPM types are just compositions (structures, arrays) of either basic or other non-basic types.
- Simple, repetitive statements – but error-prone and extensive. Luckily, it can be auto-generated for the most part.
- About 95% of the (un)marshalling code has been auto-generated out of the (PDF) specification by means of Perl, awk, and sed scripts.

Conclusion and Outlook

- TPM Emulator works with (almost) all TPM applications I know about.
- By now, about 50 out of 120 TPM commands are implemented (∼42%).
- Complete TPM Device Driver Library for the emulator according to TCG Software Stack (TSS) specification.
- Jeff Kravitz from IBM is preparing a new TPM library and some additional examples.
- Omen Wild from Dartmouth College is confident to find some students who might support the project.
- Jesus Molina from the University of Maryland is porting the emulator to a PCI embedded system. Goal is the development of a free TPM device as well as an appropriate BIOS.

Conclusion and Outlook

- TPM Emulator works with (almost) all TPM applications I know about.
- By now, about 50 out of 120 TPM commands are implemented (∼42%).
- Complete TPM Device Driver Library for the emulator according to TCG Software Stack (TSS) specification.
- Jeff Kravitz from IBM is preparing a new TPM library and some additional examples.
- Omen Wild from Dartmouth College is confident to find some students who might support the project.
- Jesus Molina from the University of Maryland is porting the emulator to a PCI embedded system. Goal is the development of a free TPM device as well as an appropriate BIOS.