Federal Office
for Information Security

Study on behalf of the
German Federal Office for Information Security (BSI)

TSS Study

Introduction and Analysis of the Open Source TCG Software Stack TrouSerS and Tools in its Environment

Version 1.0

Authors:

Marcel Selhorst, Christian Stüble, Felix Teerkorn

Sirrix AG security technologies
Lise-Meitner-Allee 4
44801 Bochum
Germany

Steering of this study:

Florian v. Samson, Daniel Nowack
Federal Office for Information Security (BSI)
P.O. Box 200363
53133 Bonn
Germany

Licensing:

This work is provided under the terms of the Creative Commons Public License by Attribution-No Derivative Works 3.0 Germany (CCPL-by-ND 3.0). In detail:

- to Share. You are free to copy, distribute and transmit the work.
- Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- No Derivative Works. You may not alter, transform, or build upon this work.

For the complete license, please see Appendix D.

Contents

1. Introduction

The relevance of “secure operating systems” has increased tremendously in recent years in many respects. This is due to the fact that many applications in both the commercial and private realm are developed for IT systems assumed to be directly connected to the Internet. Therefore, these systems and the applications running on them are continuously endangered by attacks from malware such as viruses and Trojan horses.
The latest developments in the area of Trusted Computing (TC) promise significant advances in secure operating systems, especially such enhancements in hardware and software as the Trusted Platform Module (TPM) and TCG Software Stack (TSS) technologies specified by the Trusted Computing Group (TCG). To protect an IT system, however, the availability of a TPM and a TSS is not sufficient. In addition, middleware and high-level applications must be developed to make use of the aforementioned security service enhancements.
A variety of Trusted Computing oriented software components have been released under open source licenses. The functionality and applicability of these components remain quite general, however, more specialized extensions of the basic technology are only beginning to be developed.
In order to provide an overview of existing open source software supporting Trusted Computing technology, evaluate their compliance and interoperability as well as identify shortcomings and missing pieces, this study covers the following themes:

1. Analysis of the extent to which current open source components comply with the existing specifications of the TCG. In particular, conformity to the interfaces and completeness of the implementation shall be investigated.
2. Investigation of the architecture of individual components as well as possible interactions between them in the scope of the architecture at large. Similarly, the style and programming language of the implementation is considered, which is especially important in the case of common software libraries.
3. Pointing out future Trusted Computing architectures and the interactions with existing components. Special regard is given to which components and functionalities are missing that would lead to the development of more secure applications.
4. Interoperability analysis of various security and virtualization solutions such as SELinux, Xen and Turaya, including a list of test cases covering the functionality of the components analysed in the study.

1.1 Terms and Definitions

The terms and definitions used throughout this study are defined in the glossary, while abbreviations are defined in the list of acronyms at the end of this document. Moreover, the following principal terms are used throughout:

Trusted Computing. Components and mechanisms that are compatible with the specifications of the TCG.

Linux. A term used to identify the Linux kernel, if not explicitly defined otherwise. To identify an entire operating system including the Linux kernel, the term “GNU/Linux operating system” is used.

Trusted Boot. Security property in a bootstrap architecture used according to the bootstrap model of the TCG (i.e. all software components involved in the bootstrap process are measured before execution such that remote parties can verify them).

Secure Boot. Security property in a bootstrap architecture that only bootstraps a predefined configuration of software components.If the software has been modified, the bootstrap process is interrupted.

2. High-Level Design and Component Interactions

This chapter introduces some of the main open source components of a platform enabled for Trusted Computing: the TPM Manager, a graphical TPM user management component, TrouSerS' TPM Tools, the OpenSSL TPM engine, the TrouSerS TSS, the Linux TPM Device Driver (TDD) and various TCG extensions of the GRand Unified Bootloader (GRUB) boot loader. The following sections discuss these components by describing their basic functionality and high-level design. Moreover, the interactions and the information flow between these components are illustrated.

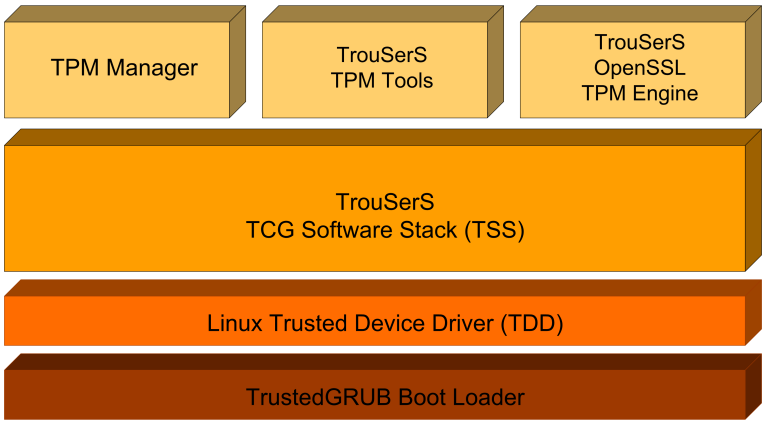


Figure 2.1.: Overview of the components described and analyzed in this chapter.

As illustrated in figure 2.1, the analyzed components constitute a layered hierarchy. Hence, the organization of this chapter is bottom-up. That is, the first three analyzed components are existing TCG extensions of the GRUB boot loader, the Linux TDD, and the TSS, followed by an analysis of the three user-space applications: the TPM Manager, TPM Tools and the OpenSSL TPM engine.

2.1. GRUB and its TC Extensions

The boot loader GRand Unified Bootloader[1] was originally developed by Erich Boleyn and is now maintained by the GNU project. GRUB has a flexible architecture and is able to boot a variety of operating systems including Linux as well as other Unix-based platforms and Microsoft Windows. It also supports the multiboot standard. Because it is free software (under the GNU General Public License (GPL)) it is an ideal base to extend with additional features.
Like nearly all boot loaders, GRUB consists of multiple stages executed in succession. The

1 http://www.gnu.org/software/grub/

first stage, `stage1`, is located in the boot sector. Its task is to establish settings essential for loading the subsequent stage (e.g. the addressing mode of the boot medium) and jump to the appropriate memory location, from where the next step in the boot process is taken. The second stage, `stage2`, is the main part of GRUB. Its task is to prepare the underlying system in such a way that the operating system kernel can be loaded. An additional feature of GRUB is to boot from a network. This feature allows the binary modules to be loaded (and even the GRUB configuration file itself) to be resident on an accessible remote platform.
The behavior of GRUB is characterized by a configuration file named menu.lst. It contains all parameters required to boot an operating system. GRUB provides a boot menu allowing users to start various operating systems (such as Linux or Windows) and/or configurations.

stage 1. GRUB `stage1` includes the starting sequence of the boot loader, which is always located in the Master Boot Record (MBR) and is activated after the BIOS has initialized the core parts of the system hardware. The size of the boot sector is always 512 bytes. However, since it also includes the 64-byte partition table, the startup code is limited to roughly 440 bytes. As mentioned above, the main purpose of `stage1` is to load the second stage and defer control to it. Since the second stage can be located on different media (e.g. a hard disk or a floppy), `stage1` has to support the necessary addressing modes. Specifically, `stage1` includes three addressing modes; two for hard drives (Logical Block Addressing (LBA) and Cylinder Head Sector (CHS)) and one for floppy disks (de facto an adapted version of CHS). After `stage2` has been located by using one of these addressing modes, its first sector (again, 512 bytes in size) is loaded into memory. finally, `stage1` jumps to the address of the memory block just loaded and defers the control over the remaining boot process to the second stage.

stage 2. `stage2` is the core part of GRUB. It loads the operating system kernel together with some additional parameters. Because the operating system kernel is usually too large to fit into the memory space of the real mode, `stage2` executes mostly in protected mode.

2.1.1. TrustedGRUB

To realize a trusted boot process, TrustedGRUB[2] extends the widely-used GRUB via security mechanisms offered by a TPM. In short, TrustedGRUB uses the Root of Trust for Measurement (RTM) functions offered by the TCG specifications to continue the chain-of-trust started by the Core Root of Trust for Measurement (CRTM) and the TCG-extended Basic Input Output System (BIOS). Additionally, TrustedGRUB generates measurement reports about the integrity of arbitrary files specified by the platform user. The main motivation for its development lays in the need for a TCG-enabled, multiboot compliant

2 http://TrustedGRUB.sourceforge.net/

boot loader for the Turaya security framework[3] . However, TrustedGRUB is able to boot all operating systems supported by GRUB.

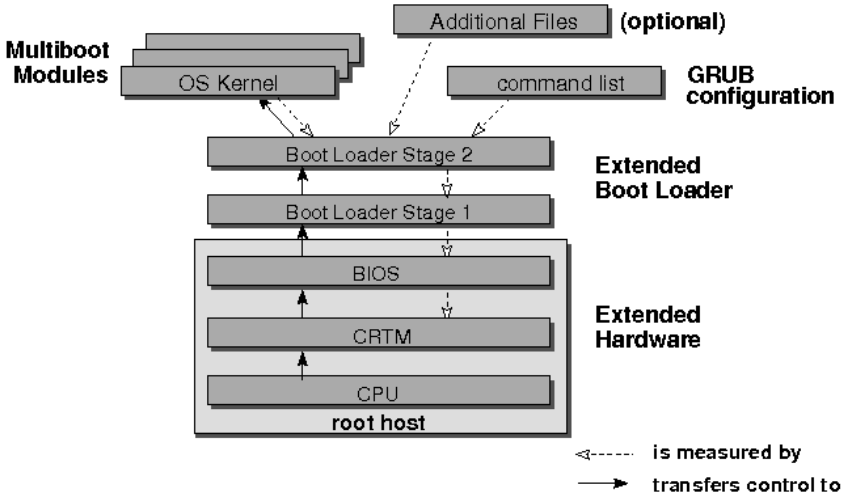


Figure 2.2.: Components involved in the bootstrap process of a TCG-extended PC platform.

Figure 2.2 outlines the steps implemented by TrustedGRUB. It shows the control flow between the BIOS, the GRUB stages, and the operating system kernel. The main difference introduced by the TCG is the CRTM measuring the BIOS, and the BIOS measuring the boot sector. In addition, TrustedGRUB measures relevant sections of its configuration file, an optional list of files, and the multiboot modules to be loaded.
TrustedGRUB extends `stage1` so that it measures the first sector of `stage2`. A problem here is the limited size of stage1. In the current version of TrustedGRUB, the solution to this problem is to remove some disk addressing modes, resulting in two different versions of stage1 that supports booting from either a floppy or a hard disk.
In order to guarantee the integrity of the boot process, TrustedGRUB has to measure all input used by `stage2` including the kernel, further multiboot modules and configuration information. The default configuration options of GRUB are set in the configuration file, menu.lst, loaded by `stage2`. However, users can change or extend the configuration options using the command shell provided by GRUB. The directives in the configuration options are not carried out immediately; instead, they are stored in a list and not executed until the user has selected the configuration to be booted.

3 http://www.emscb.de/

2.1.1.1. Installation and Configuration

In order to install TrustedGRUB, download the source code distribution from the project page located at http://sourceforge.net/projects/TrustedGRUB/. Extract and compile the archive using the build script below:

Listing 2.1: Configuring and compiling

```
# tar xzf TrustedGRUB-<ver >. tgz
# cd TrustedGRUB-<ver>
# . / build_tgrub . sh
# make install
```

Additional parameters for the command `./build_tgrub.sh` can be listed with the option `--help`. The installation of TrustedGRUB can be peformed manually:

Listing 2.2: Installing TrustedGRUB into the MBR

```
 # cp stage1 / stage1 / boot / grub
 # cp stage2 / stage2 / boot / grub
 # . / grub / grub
 #   root (hdX,Y) // boot partition
 #   setup (hdX)  // hard disk to install TrustedGRUB on
 #   quit
Alternatively , use the grub install utility
located in the 'util ' subdirectory :
 # cd util
 # chmod a+x grub-install
 # grub-install / dev / hdX
     # tar xzf TrustedGRUB-<ver >. tgz
     # cd TrustedGRUB-<ver>
     # . / build_tgrub . sh
     # make install
```

2.1.1.2. Additional Features

TrustedGRUB comes with additional features, described below:

Password Check. TrustedGRUB provides an additional feature that asks for a user password if the parameter `--with-password-dialog` is added to a module. It prompts for a password and replaces this parameter by the string `password=<yourpassword>`. This feature provides the basis to realize pre-boot authentication mechanisms as required by some microkernel-based projects.

Checkfile. TrustedGRUB can measure the integrity of arbitrary files specified by the user. This is done by introduction of the new GRUB command checkfile(). By employing this either in menu.lst or by command line, the user is telling TrustedGRUB where in the file system a checkfile can be found. The checkfile contains a list of tuples including file names and corresponding SHA-1 hash value references.
Generally, the checkfile option allows the integrity of all files on a system to be checked. In contrast to the measurement of the multiboot modules to be loaded, TrustedGRUB compares the measurement results with the hash values in the checkfile and, in the case of an error, it interrupts the boot process to prevent manipulated components from doing harm to the system. The syntax of the checkfile is as follows:

```
checkfile (hd?,?)/somewhere/check.file
```

Make sure that the drive parameter (hd?,?) and the path are configured correctly; otherwise TrustedGRUB is not able to boot! The checkfile itself contains a list of tuples of arbitrary length (however, the checkfile MUST NOT be larger than 8192 bytes) with a well-defined syntax as follows:

```
2647eeae7290c5a58dacb87347ba074de7e47bac (hd0,1)/etc/passwd
a97fbdba48d4a6340baff683941079dde56044e0 (hd0,1)/etc/shadow
d0fc1992068b660271af7bae7a1dea4258ef0b8b (hd0,1)/etc/group
bdf2a7d6132fe5ddf164b3bdae8764ab7433359a (hd0,1)/etc/fstab
```

The first component is a 40-byte alphanumeric value marking the SHA-1 hash value of the succeeding file (the value can be created either by sha1sum under GNU/Linux or the program create_sha1 that comes with the TrustedGRUB distribution) followed by a single white space character. The second component has to be the absolute path (including the drive reference) of the file corresponding to the hash value followed by a new line character.
The integrity of all files listed in this checkfile is verified during system boot by comparing the referenced hash values to newly-computed values. If some of these do not match, a warning is displayed offering the option of either continuing with the booting of a potentially contaminated system or aborting the boot process completely. This is an initial step towards secure boot as defined in section 1.1. All files verified by the checkfile option are extended into the TPM's Platform Configuration Register (PCR) 13.
The integrity of the checkfile itself is currently not verified by TrustedGRUB. In case an attacker replaces a file on the filesystem and replaces the corresponding hash value inside the checkfile, TrustedGRUB will not recognize this manipulation during the boot process, since the measured hash value of the file is equal to the reference hash value inside the checkfile. Certainly, the hash value of the modified file will be stored inside PCR 13 of the TPM, such that a manipulation of the checkfile will always result in a modified value of PCR 13. To implicitly guarantee data or system integrity, it is therefore necessary, to seal data to PCR 13. The manipulation will then automatically be detected as soon as one tries to unseal the data.

Component	GRUB	TrustedGRUB
stage1	239	297
stage2	19380	22805
lib	1479	1482
util	1104	2220
grub	1103	1115

Table 2.1.: Comparison of the number of code lines of GRUB and TrustedGRUB components.

sha1 Utility. This new GRUB utility simply measures the SHA-1 hash of the given file and prints the result. The syntax of sha1 is as follows:

```
sha1 (hd?,?)/somewhere/hash/my/file
```

2.1.1.3. Quality of the Implementation

Code Complexity and Quality. The majority of the GNU GRUB boot loader is written in ANSI C (85%). However, some portions had to be written in assembler in order to fit into the limited memory space available (e.g. stage1 into the 512-byte MBR). In order to extend the new TPM functionality and turn GRUB into TrustedGRUB, about 5,400 lines of additional code were necessary. Table 2.1 lists the various extensions.
Due to the complexity of the boot process and the large amount of file systems and different boot media, GRUB itself is a very complex system. Moreover, GRUB offers many features which increase usability while expanding the code and, hence, its complexity. All extensions made for TrustedGRUB include extensive documentation in the source code.

Documentation and Support. A manual is included with the TrustedGRUB distribution. Additionally, an enhanced Wiki and issue-tracking system (TRAC) is available for TrustedGRUB[4] as well as a mailing list hosted at the sourceforge project page[5] .

Limitations. TrustedGRUB has the following known limitations:

- On some TPM BIOS implementations, TrustedGRUB does not run if the TPM is disabled in the BIOS. As of this writing, this has only occurred on the IBM Thinkpad T41p.
- On some notebooks, TrustedGRUB is unable to extend the PCR of the TPM due to the lack of a `TCG_PASS_THROUGH_TO_TPM` feature in the BIOS. However, this issue is not related to TrustedGRUB but to a wrong CRTM.
- floppy support is currently not included due to the limited size of GRUB `stage1`.

4 http://projects.sirrix.com/trac/trustedgrub
5 http://sourceforge.net/projects/TrustedGRUB

2.1.1.4. Future Work

The following improvements to `stage2` of TrustedGRUB are currently planned:

- Support of a configuration option to define the PCRs to be extended.
- Support of property-based attestation/sealing to enable the binding of data to a public key instead of binary measurements, as described in [11].

2.1.2. GRUB-IMA

GRUB-IMA is another TC extension for the commonly-used GRUB boot loader. GRUB-IMA has been developed by IBM for their Integrity Measurement Architecture (IMA). GRUB-IMA measures all GRUB components and OS components loaded by GRUB. After measuring the loaded files, the hash results are extended into the PCRs as described in the Table 2.2. Additionally, the measurement values are stored inside the ACPI measurement log using the event types defined in Table 2.3.

PCR	Usage
4	MBR information, `stage1` and stage2
5	ACTIONs (here: EventLog)
8	OS components (kernel, initrd, module, `measure`-command)

Table 2.2.: PCR usage within GRUB-IMA.

Event-type	Description
0x1005	SHA1 digest of GRUB ACTION message
0x1105	SHA1 digest of kernel commandline strings
0x1205	SHA1 digest of kernel image
0x1305	SHA1 digest of initrd image
0x1305	SHA1 digest of module image

Table 2.2.: Event-types of GRUB-IMA.

2.1.2.1. Installation and Configuration

GRUB-IMA can be downloaded from the project page of the TrouSerS project hosted on sourceforge.net[6] . The GRUB-IMA package requires the presence of the original GRUB sourcecode, since it is only provided as a patch. Listing 2.3 shows the required steps to apply the GRUB-IMA changes to the sourcecode of GRUB:

6 http://sourceforge.net/projects/trousers/

Listing 2.3: Installing GRUB-IMA

```
      # rpm -ivh grub -0.97 -13. src . rpm
      # cd /usr/src/redhat/SPECS /
      # rpmbuild -bp grub.spec
      # cd ../BUILD/grub-0.97/
      # patch -p1 < grub-0.97-13-ima-1.1.0.0.patch

Build:
      # autoreconf --install --force
      #CflAGS=-Os  -g  -fno-strict-aliasing  -Wall  -Werror  \
        -Wno-shadow -Wno-unused -Wno-pointer-sign
      # export CflAGS
      #./configure  --sbindir=/sbin  --disable-auto-linux-mem-opt  \
        --enable-ima
      # make

Install GRUB-IMA (this step does not update the GRUB in MBR):
      # make install

final installation:
      # /sbin/grub-install install_device
```

2.1.2.2. Additional Features

GRUB-IMA offers some new commands in the GRUB command shell and in the GRUB configuration file described in the following subsections.

tpm. Inside the GRUB command shell, one can execute the new `tpm` command as shown in Listing 2.4

Listing 2.4: tpm command in GRUB-IMA

```
tpm pcrs     shows values of PCR registers
tpm eventlog shows eventlog measured by BIOS and GRUB
tpm test     test TCGBIOS Int 1 AH calls
```

measure. In addition, the `measure` command can be used to measure any OS files. This is done in the GRUB configuration file by adding the line measure filename. The results of the measurement will be extended into a PCR. Listing 2.5 shows an example `grub.conf`.

Listing 2.5: measure command in GRUB-IMA

```
title CentOS (2.6.18-8.1.4.el5)
      root (hd0,0)
      measure /etc/selinux/config
```

```
measure /etc/selinux/targeted/policy/policy.21
kernel /boot/vmlinuz-2.6.18-8.1.4.el5 ro root=/dev/sda1 rhgb
initrd /boot/initrd-2.6.18-8.1.4.el5.img
```

2.1.2.3. Quality of the Implementation

Code Complexity and Quality. The GRUB-IMA patch comprises about 113 kB of code, consisting mostly of assembler routines. The source code extensions are documented extensively in the source code.

Documentation and Support. GRUB-IMA includes neither documentation, wiki nor forum. Only a short README file is included in the source package. Support is available from the TrouSerS mailing list on the project page. The TrouSerS mailing list holds for GRUB-IMA as well.

Limitations. Since the SHA-1 calculation is done using the TPM hardware chip, the measurement process is very slow. As well, GRUB-IMA is incapable of running microkernels (e.g. L4) directly, therefore it cannot be used with architectures like Turaya.

2.1.3. OSLO

Open Secure LOader (OSLO) is an open source boot loader that uses the SKINIT instruction available in newer AMD64 processors [2]. OSLO requires a version 1.2 TPM and can be used for secure booting by creating a dynamic root of trust for measurement. It is a replacement for the static root of trust provided by the BIOS and subsequent secure bootloaders, which are used on machines not supporting SKINIT. OSLO is booted by a multiboot loader (e.g. GRUB). Afterwards, OSLO initializes the Dynamic Root of Trust for Measurement, hashes all loaded files and extends the result into the dynamic PCR's of a v1.2 TPM. In order to communicate with the TPM, OSLO is shipped with a TPM TIS device driver.

OSLO is splitted into three parts:

1. OSLO: the main programming
2. Beirut: A helper program that hashes the command line of other multiboot modules.
3. Pamplona: A helper program that does everything to reverse the steps done by OSLO. For example, it removes the device-protection (DEV-flag) and clears the global interrupt flag. It allows one to use OSLO but start an unmodified OS in an insecure fashion afterwards.

2.1.3.1. Installation and Configuration

OSLO can be downloaded from http://os.inf.tu-dresden.de/~kauer/oslo/. Since OSLO forms a new root of trust by using the Dynamic Root of Trust for Measurement of the TPM and the new AMD features, it is not necessary to start from a secure boot loader. Listing 2.6 shows how OSLO is started with TrustedGRUB as it is used in the OpenTC project[7] . One can see that the OSLO boot loader is handled like a multiboot kernel and therefore takes responsibility for loading and measuring all of the modules following.

2.1.3.2. Quality of the Implementation

Code Complexity and Quality. OSLO has roughly 1000 lines of code. The compiled binary has a size of 4 kB.

Documentation and Support. A short manual is available in the source package. Ad- ditionally, a mailing list is hosted by the Dresden Operating System group (as needed for their L4 kernel implementation) located at http://os.inf.tu-dresden.de/mailman/listinfo/l4-hackers/.

Limitations. Currently, at least two bugs are known: The device initialization is missingand a ”type of memory” bug is present.

2.1.4. Bootstrapping

The word bootstrapping is usually abbreviated by the more common word booting. Booting is the necessary process of initializing the main hardware components (usually performed by the BIOS) and afterwards starting the operating system of the computer (usually carried out by the boot loader). This section describes the dynamic behavior of a platform during bootstrapping with TrustedGRUB.
The integrity of a platform can only be ensured if the chain-of-trust is not interrupted. The anchor of this chain is the combination of the CRTM (in a modern PC, this is an extension of the BIOS) and the TPM. A valid trust chain, at the minimum, is necessary in the following scenarios:

a) To ensure the integrity of the current platform while only revealing certain data in case the system is unmodified and in the same state as intended.

7 http://www.opentc.net

b) To attest the configuration of the current platform to an external entity. For instance, a company allowing access to its internal network only if the connecting PC of a telecommuting employee fulfills certain security requirements.

Since the trust of the entire platform relies on every link of the chain leading to the root (of the chain), one has to ensure that the chain is not interrupted.

Required Steps. The following steps are necessary to build a valid chain-of-trust upon booting (refer to figure 2.3).

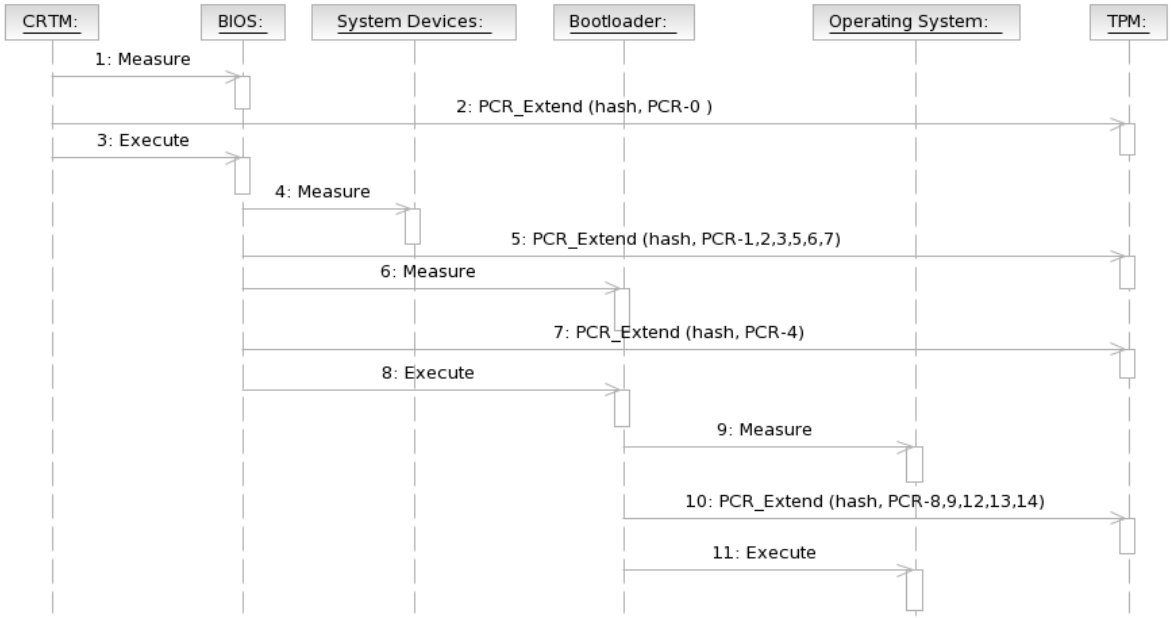


Figure 2.3.: The Chain of Trust with a TCG-enabled Bootloader.

Listing 2.6: TrustedGRUB booting OSLO

```
# OTC - Proof of Concept prototype
# L4 boot.lst - for tGRUB 1.1.0 or newer versions
#
hiddenmenu
default=0
timeout=3
#
#OTCGRUBROOT, OTCL4 and LINUXROOT variables are set in the calling
#common menu.lst
#Other variables are set in the calling menu.lst specific for L4

title         OTC : booting L4 . . .
root          $ (OTCGRUBROOT)
kernel /boot/oslo/oslo
module /boot/oslo/beirut
module /boot/oslo/pamplona
module $(OTCSPEC)/bin/bootstrap
modaddr 0x02000000
module $(OTCSPEC)/bin/fiasco -nokdb -nowait
module $(OTCSPEC)/bin/sigma0 -v
module $(OTCSPEC)/bin/roottask -configfile -sigma0 task modname
       "bmodfs" attached modules task modname "compmgr" module
module $(OTCSPEC)/etc/roottask.config
module $(OTCSPEC)/bin/names
module $(OTCSPEC)/bin/events
module $(OTCSPEC)/bin/dm_phys -e -v --isa=0x800000
module $(OTCSPEC)/bin/simple_ts -e -t 200
module $(OTCSPEC)/bin/rtc -v
module $(OTCSPEC)/bin/l4io
module $(OTCSPEC)/bin/bmodfs
          module $(OTCSPEC)/bin/libld-l4 . s . so
          module $(OTCSPEC)/bin/libloader . s . so
          module $(OTCSPEC)/bin/vmlinuz-dom0
          module $(OTCSPEC)/bin/vmlinuz-domUT
          module $(OTCSPEC)/etc/$(DOM0)
          module $(OTCSPEC)/etc/$(DOMT)
          module $(OTCSPEC)/etc/$(DOMU)
          module $(OTCSPEC)/etc/$(DOMS)
          module /boot/preroot.img
          module $(OTCSPEC)/bin/loader
module $(OTCSPEC)/bin/mgui -r 1024 x768
module $(OTCSPEC)/bin/compmgr
module $(OTCSPEC)/etc/$(COMPMGRCONF)
module $(OTCSPEC)/bin/compmgrclientl4 -f BMODFS -t loader -l
       $(DOM0) $(COMPMGRGUESTS)
```

1. CRTM: The CRTM measures the BIOS and extends the results into the TPM.
2. BIOS: The BIOS measures all firmware before execution (e.g. Option ROM of the graphic adapter) as well as the boot loader (stored in the MBR of the boot media) and extends the results into the PCR s of the TPM.
3. Bootloader: The bootloader continues the trust chain as the boot loader by measuring all operating system files that have been loaded and extending the results into the TPM. Additional files that need to be included in the chain-of-trust can be measured and extended with the checkfile-function as described in Section 2.1.1.2.
4. OS Kernel: At this point of the trust chain, the operating system has to ensure that the chain is continued in order to keep the platform configuration current, even if the system changes upon execution.

After bootstrapping, the operating system should have been loaded, all executed code measured and extended into the TPM. The platform configuration is now reflected by the values of the PCR s inside the TPM and is now ready to verify its own integrity or attest it to an external entity. Since the TCG definition of TrustedBoot ends immediately following the bootstrap process, it is in the responsibility of the operating system, now running, to continue the chain-of-trust. Several approaches are currently under research and/or implementation. IBM, for example, has implemented the mentioned IMA extension for Linux[8] .

2.2. Linux TDD

In order to use a TPM after Linux has been bootstrapped, a TPM device driver is necessary. Since the TPM is implemented as a hardware module, the device driver for the specific TPM has to be loaded into Linux in order to access the hardware. According to the TPM specifications, this abstraction layer is named TPM Device Driver. The following sections describe the current state of TPM support in Linux.

2.2.1. High-Level Design

Generic TPM interface. In the early stages of the TPM specification, no assumptions were made regarding the design of TPM interfaces in the operating system[9]. It was necessary for the vendor to define the communication behavior with their chip. Furthermore, various TPM vendors might provide different communication interfaces, making it very hard for application developers to support every type of TPM chip on the market. In order to circumvent this problem, IBM developed a standardized interface for Linux, making the communication independent from the actual TPM chip used on a specific system. This helps application developers, since only one way of communicating with the TPM has to be supported.

8 http://domino.research.ibm.com/comm/research_people.nsf/pages/sailer.ima.html
9 Meanwhile, this has changed due to the introduction of the TSS specification [7].

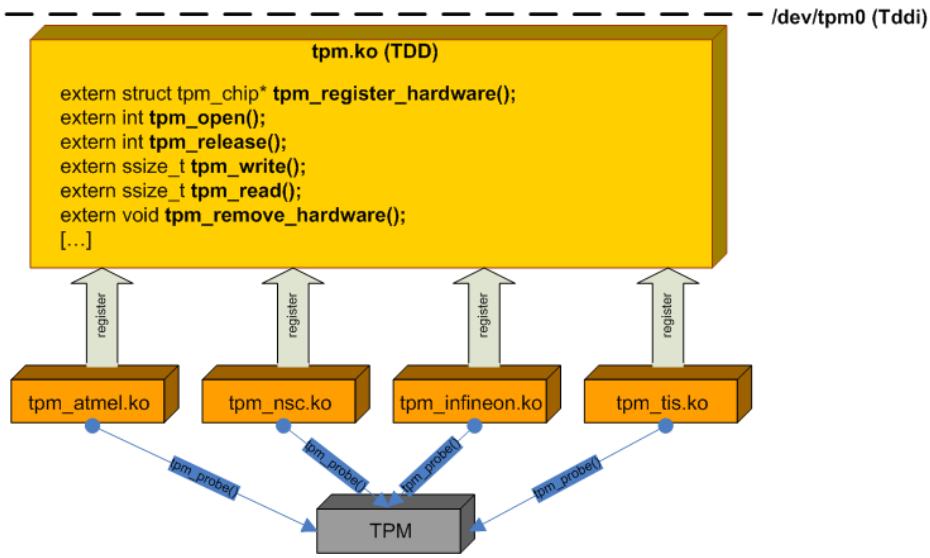


Figure 2.4.: The TPM interface inside Linux.

This TPM interface has been completely integrated into Linux, allowing even more than one TPM chip at a time. In order to communicate with the TPM, the kernel provides bidirectional access to the TPM interface via the Linux device file system. The first TPM chip on the system is bound to `/dev/tpm0`; the second chip would be bound to `/dev/tpm1`, and so on. All commands which are written to or read from the device (e.g `/dev/tpm0`) have to be TCG TPM commands as defined in [24] and [27]. figure 2.4 shows the internal structure of the Linux TDD architecture and its vendor-dependent components.
In addition to the `device` file system used for communicating with the chip, more information on the device is available via the `sys` file system located under `/sys/class/misc/tpm0`[10]. The `sys` file system entries for the TPM offer a more convenient way of gathering information concerning the used chip (e.g. the current content of PCRs, the public key of the TPM or various capabilities) as well as a means to cancel a current operation. Listing 2.7 shows how to read the current PCRs of the TPM and Listing 2.8 shows how to cancel the current operation.

Vendor-specific TPM drivers. Since the external view of the TPM interface is independent from the actual TPM chip implementation, it is necessary to provide vendor-specific device drivers responsible for communicating with the chip. These device drivers are not directly accessible from the outside, since they are only used internally within Linux. All Version 1.1b TPMs lack requirements on the communication process, leading to the necessity of various vendor-specific TPM device drivers for various TPM implementations. The following TPM 1.1b device drivers are currently available within Linux:

- Infineon TPM 1.1b
  `modprobe tpm_infineon`
- Atmel TPM 1.1b
  `modprobe tpm_atmel`

10 replace the number with the corresponding chip number

- NSC TPM 1.1b
  `modprobe tpm_nsc`

Generic TPM v1.2 driver. With the introduction of the TCG specification 1.2 [24], a standardized TPM communication interface called TPM Interface Specification (TIS) was specified in [23]. The corresponding implementation of the specification is also available within Linux and supports most of the TPM v1.2 chips currently available.

- TPM 1.2 TIS
  `modprobe tpm_tis`

Figure 2.5 shows how to select the TPM device drivers inside Linux.

Listing 2.7: Content of PCRs via sysfs

```
# cat /sys/class/misc/tpm0/device/pcrs
PCR-00:BF D3 08 05 08 E7 98 D7 D2 CB 0E 15 A8 B8 74 DB 8F F8 CB A2
PCR-01:79 50 52 AB EB DB 2A 1B C7 73 0E 59 3D BA B4 6F CE 8C BE DE
PCR-02:EB B3 BA AE E7 57 4B B6 37 AA AB 67 0F 9A C1 BC EB 6F 80 F3
PCR-03:04 FD EC DD 50 1D AF 0F 62 4C 1F 99 60 12 CF 30 44 ff 46 10
PCR-04:CF C7 FA 94 66 30 B5 0C AF 55 44 F1 96 C1 04 4B 20 99 E6 98
PCR-05:04 FD EC DD 50 1D AF 0F 62 4C 1F 99 60 12 CF 30 44 ff 46 10
PCR-06:05 13 FA E7 AE 00 59 0B 4A 98 A3 E1 86 BD F0 C6 1C 2C 80 AA
PCR-07:04 FD EC DD 50 1D AF 0F 62 4C 1F 99 60 12 CF 30 44 ff 46 10
PCR-08:C1 EC 2A 49 F3 43 F5 A2 9E A6 A4 0C A6 82 36 11 DB 53 F9 5F
PCR-09:0B 47 F0 9C FA 1F 0E D8 E7 8D EA 5C 27 32 54 E9 CD BC 5F 80
PCR-10:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-11:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-12:3C D6 6E 89 17 4E CD F1 15 D9 BA D3 C7 5E 0F BF 9C A2 ED B0
PCR-13:6B C9 23 12 C7 32 36 90 ff 7D 5A D0 E0 D2 A6 00 29 9B 10 B0
PCR-14:39 21 82 94 E8 D5 0A B4 2E 3E 84 64 CC 41 3C 40 AB CA B5 0A
PCR-15:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Listing 2.8: Canceling the current TPM operation via sysfs

```
echo 1 > /sys/class/misc/tpm0/device/cancel
```

2.2.2. Quality of the Implementation

The Linux TPM interface and the device drivers work as intended. Due to the review process required for integration into the kernel, the source code is up-to-date, maintained and of high quality (albeit not documented very well). The TPM TIS 1.2 implementation complies with the [23] specification. Currently, no manual is available explaining how to

use the TPM within Linux, but support is available from a mailing list[11] .

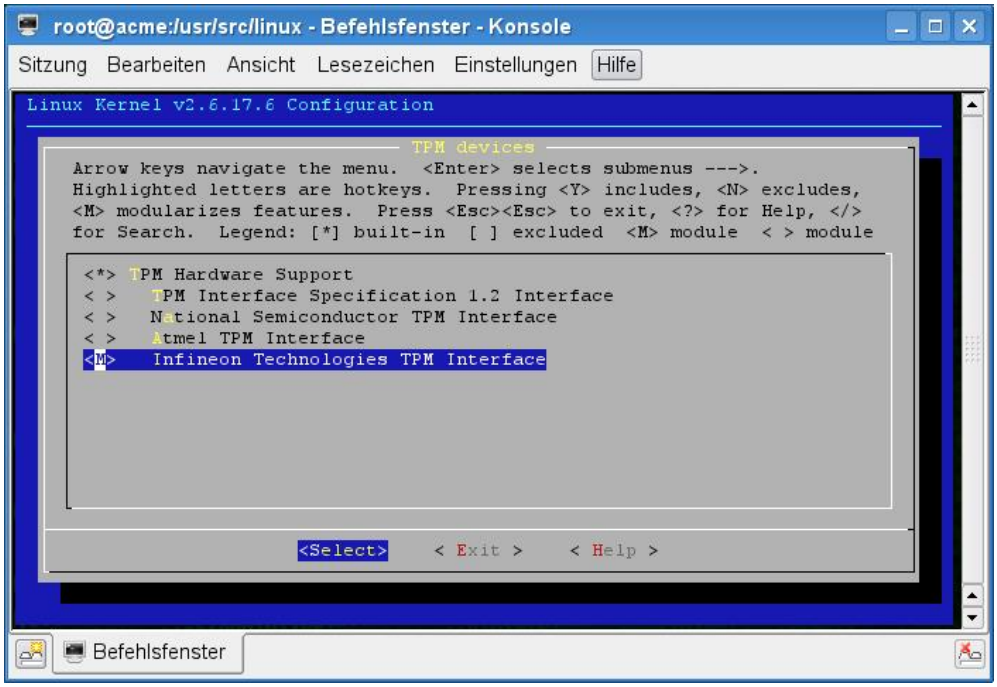


Figure 2.5.: TPM device driver selection inside Linux.

2.3. TrouSerS TSS

The TCG Software Stack provides an Application Programming Interface (API) to operating systems and applications so that they can use the functionality provided by a TPM. It allows different applications and even different operating systems, running as virtual machines on top of a Virtual Machine Monitor (VMM), to operate with each other, as long as they comply to the TSS specification.
The main purpose of a TSS is to multiplex access to the TPM, since a TPM has limited ressources and can only communicate with one client. Moreover, not all functions related to Trusted Computing require TPM access. These functions are located within the TSS.
TrouSerS is an open source TSS implementation maintained by IBM since 2004. The latest version released is stable and implements TSS version 1.1 [6] interfacing version 1.1b TPMs. Since the TCG has released a new version of the TSS specification [7] to support new functionality provided by version 1.2 TPMs, a new version of TrouSerS is currently under heavy development. At the moment, some of the functionality required for version 1.2 compliance has yet to be made available. The current progress can be tracked at [9].

11 https://lists.sourceforge.net/lists/listinfo/tpmdd-devel/

2.3.1. High-Level Design

The TSS is comprised of three layers. figure 2.6 shows a TSS instance.

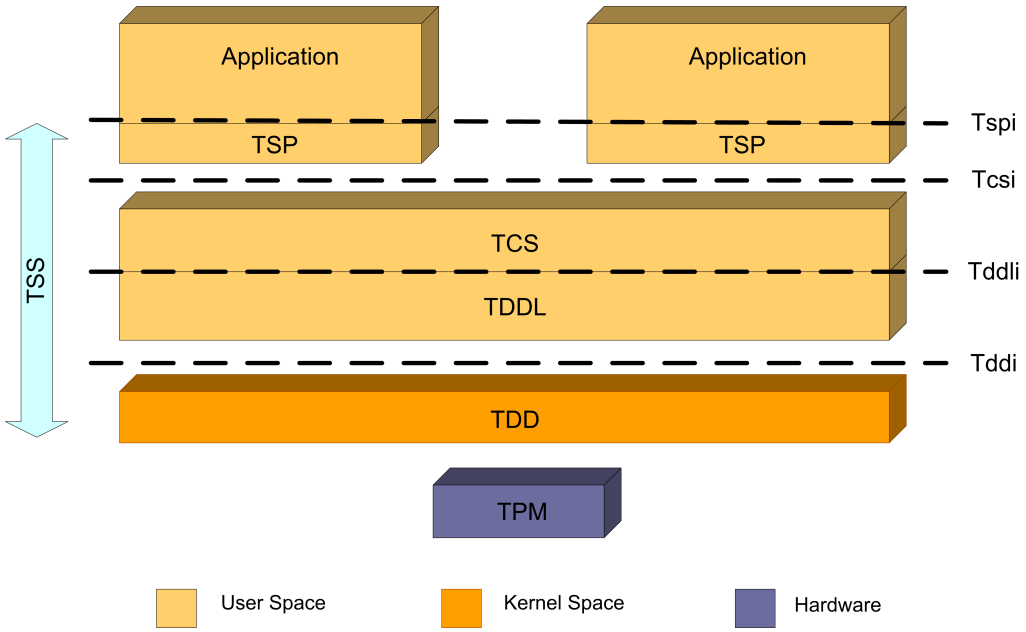


Figure 2.6.: Example object model of a TSS instance.

TSP and TSPI. Every application employing TPM functionality has to use a TSS Service Provider (TSP) via its interface, the TSS Service Provider Interface (TSPI). Every application uses its own TSP, which is loaded as a library into any application process making use of it. The TSP provides high-level TCG functions as well as auxiliary services such as hashing or signature verification. The TSP is a must for every TSS implementation. The TSP can be a discrete module; it is possible to integrate the TSP into other platform modules. One aspect of the TSP is the TSP Context Manager (TSPCM), which provides dynamic handles allowing efficient usage of TSPs and application resources. A second aspect is the TSP Cryptographic Functions (TSPCF), which are provided to make full use of the protected functions in a TPM. The TSPI is defined as a C interface.

TCS and TCSI. The TSS Core Service (TCS) provides a common set of operations to all of the TSPs running on a platform. Since there is only one instance of a TCS per TPM, the main task of the TCS is to multiplex TPM access. Moreover, the TCS provides operations to store, manage and protect keys as well as privacy-sensitive credentials associated with the platform. The TCS is implemented as a user space daemon.

TDDL and TDDLI. The TCG Device Driver Library (TDDL), accessed via the TCG Device Driver Library Interface (TDDLI), provides a unique interface to different TPM driver implementations. Moreover, it provides a transition between kernel mode and user mode[12] . The TDDL provides functions (e.g. Open, Close, GetStatus) to maintain communication with the TDD. Additionally, it provides functions (e.g. GetCapability, SetCapability) to get and set attributes of the TPM, TDD and TDDL as well as direct functions (e.g. Transmit, Cancel) to transmit and cancel TPM commands.

TDD and TDDI. The TPM Device Driver incorporates code that understands the specific behavior of the TPM. It is the only module that communicates directly with the TPM. It receives byte streams from the TDDL and sends them on to the TPM, returning its response back to the TDDL. Although the TSS specification mentions the TDD interface TDDi, the current TSS specification does not define it. different from the TDDL, which is implemented in user space, the TDD is implemented in kernel space.

2.3.2. TSS and Linux TDD

In the following, the interfaces involved in the communication between TSS and Linux-TDD are described.

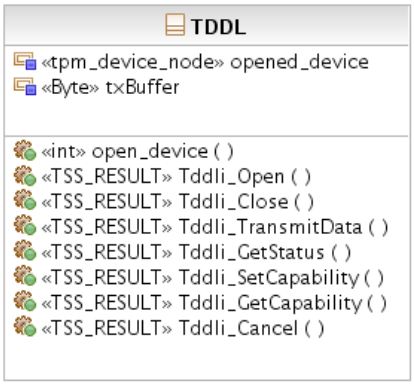


Figure 2.7.: The interface of the Trusted Device Driver Library (TDDL).

12 The current TSS design assumes a monolithic operating system including device drivers running in kernel mode.

In order to exchange data between the TrouSerS and the TPM, TrouSerS makes use of its internally built-in TDDL, which is responsible for the communication with the Linux TPM device drivers (as described in Section 2.2). The main functions offered by the TDDL are specified in the TSS 1.2 API.

Since the communication between the TSS and the Linux TDD is handled within the TDDL, all necessary functions are implemented in a library called libtddl.a, which is only used by the TCS daemon. The source code of this library is located inside the TrouSerS code base under `src/tddl/tddl.c`.

Upon starting TrouSerS, a TCS daemon is loaded that first tries to identify the availability of a TPM in the system. This is done by looking for certain device nodes on the device file system. In detail, the `Tddli_Open()`-function looks for `/dev/tpm`, `/dev/tpm0` and `/udev/tpm0`. If found, the device is opened and thereby blocked for any other application trying to use this specific TPM.
In case an application uses the TSS, a byte buffer is generated that contains the TPM function call including all necessary parameters to carry out the request. The tcsd then communicates with the TPM device driver using the `Tddli_TransmitData() function`. This sends the data in the buffer to the corresponding TPM device driver device node on the `device` file system and waits for the TPM device driver to complete the request. After the TPM device driver has passed the data to the TPM, it places the response from the TPM on the same device node, which the `Tddli_TransmitData()` function reads and returns the response. During termination of the `tcsd`, the function `Tddli_Close()` unlocks the TPM device driver. figure 2.8 describes the communication sequence between TrouSerS and the TPM.

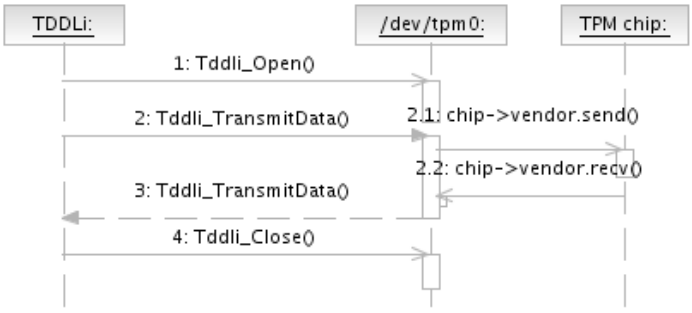


Figure 2.8.: Communication between the TrouSerS TDDI and the TPM.

2.3.3. Installation and Configuration

Installation. The latest stable release version 0.3.1 of TrouSerS was published at the beginning of November 2007 and can be downloaded at
http://sourceforge.net/projects/trousers/.
The development branch of TrouSers is managed via CVS. To download the current version via CVS, the following commands have to be executed in a shell:

```
# cvs -d:pserver:anonymous@trousers.cvs.sf.net:/cvsroot/trousers \
  login
# cvs -z3 \
  -d:pserver:anonymous@trousers.cvs.sf.net:/cvsroot/trousers co  \
  -P trousers
```

In the newly-created `trousers` folder, the implementation can be compiled[13] using the following commands:

```
# sh bootstrap.sh
# ./configure
# make
  as root
# make install
```

TSS 1.1	TSS 1.2
80% ANSI C	91% ANSI C
43,000 lines of code	56,000 lines of code

Table 2.4.: Number of code lines of TrouSerS version 1.1 and 1.2.

After successfully installing TrouSerS, the TCS daemon `tcsd` is started by entering the command: `startproc –u tss /usr/local/sbin/tcsd`.

Configuration. TrouSerS comes with a configuration file located under `/etc/tcsd.conf`. Additionally, TrouSerS stores its TPM secrets in the file `/var/lib/tpm/system.data`.

2.3.4. Quality of the Implementation

Due to the high complexity and the elaborate specification, the implementation of a TSS stack is very challenging. The source code itself is implemented in ANSI C and documented extensively.

13 Please note that TrouSerS can only be compiled with GCC v4.1

Test cases. The TrouSerS developers provide a set of around 450 test cases in their CVS for their v1.1 and v1.2 TSS implementations. To download the test suite, check out the following repository:

```
# cvs -d:pserver:anonymous@trousers.cvs.sf.net:/cvsroot/trousers \
  login
# cvs -d:pserver:anonymous@trousers.cvs.sf.net:/cvsroot/trousers \
  co -P testsuite
```

Documentation and Support. TrouSerS offers various mailing lists for both developers and users. A set of FAQs is available online[14] . The source code package comes with a `doc` directory containing low-level design documents of the implementation. A manual for most of the commands is available, too.

2.4. The TrouSerS TPM Tools

The TrouSerS project offers the TPM Tools, a set of programs to maintain TPM's. The TPM Tools provide commands allowing a platform administrator to manage and diagnose the TPM resident on a platform. In addition to this, the package contains commands for making use of the capabilities available in the TPM PKCS#11 interface implemented in the openCryptoki project, a PKCS#11 implementation for Linux. The TPM Tools package is released under the Common Public License.

2.4.1. Installation and Configuration

It is possible to download the TPM Tools package from the TrouSerS project site under http://sourceforge.net/projects/trousers/. Clicking the 'Download trousers' button should expose the link tpm−tools. To unzip the file, enter the following command in a shell:

```
# tar xzf tpm-tools-1.3.0.tar.gz
```

After extracting the archive, change to the newly-created tpm−tools folder, then enter the following commands:

```
#  sh ./bootstap.sh
#  ./configure
#  make
as  root
#  make install
```

14 http://trousers.sourceforge.net/faq.html

2.4.2. High-Level Design

This section describes the connection between user space applications and the TSS using the TPM Tools as an example.
The TSS is accessed through the TSP interface implemented by the library `libtspi.so`. Although this library is implemented in ANSI C, it is based on an object-oriented design. Therefore, all TSPI functions deal with one or more references addressing certain instances of a class. The TSPI, as provided by the libtspi.so, defines the following classes shown in figure 2.9.
The main class is Context which represents a connection to a TCS running on either a local or remote TCG computing platform. The methods provided in this class manage resources, memory and other objects. Another focus is to provide connections to a TCS and a persistent storage database. The interaction between TCS and TSP under Linux is realized by TCP sockets, where the TCS daemon listens on port number 30003. Most functions are defined in the TPM class, which represents the owner of a TPM and provides some basic control and reporting functions. The `Policy` class can be used to configure policy settings. The `Key` class offers TCG key handling and functionality.

2.4.3. Functionality

In the current version, TPM Tools supports seventeen command line tools to maintain the TPM, including the following functions:

- owner management
- endorsement key management
- configuration management

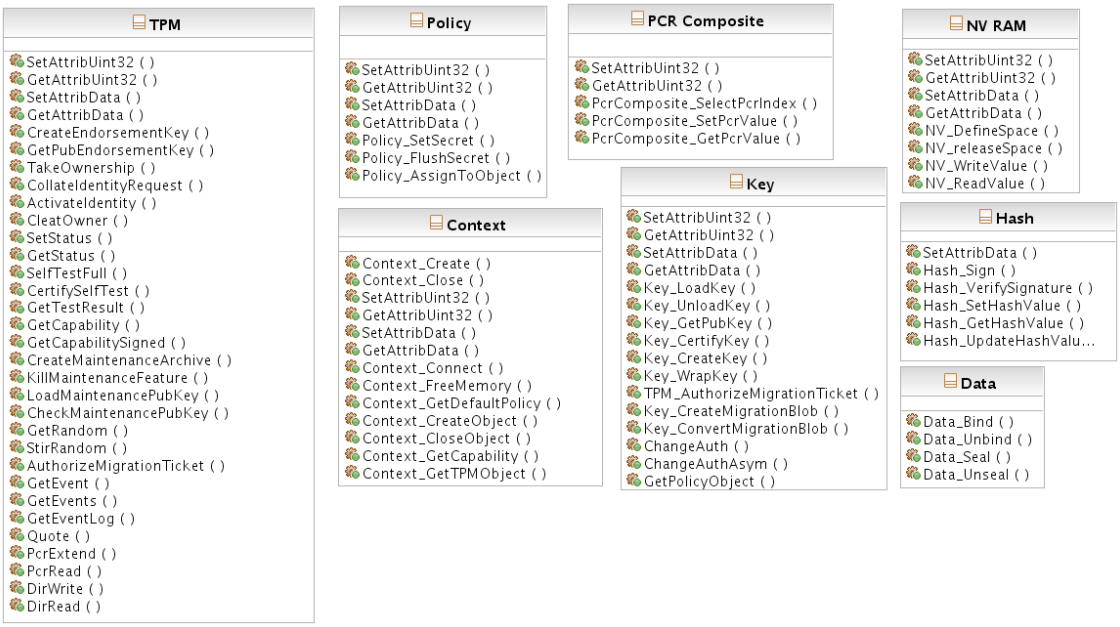


Figure 2.9.: Classes defined by the TSP interface (TSPI).

Command	Description
tpm_changeownerauth	Change the authorization data associated with the owner or SRK
tpm_clear	Return the TPM to the default state (unowned, disabled, inactive)
tpm_createek	Create an Endorsement Key pair in the TPM
tpm_getpubek	Display the public portion of the Endorsement Key in the TPM
tpm_resetdalock	Reset the dictionary attack lock for the user (requires owner authentication)
tpm_restrictpubek	Restrict the ability to display the public portion of the Endorsement Key to the owner
tpm_revokeek	Revoke the Endorsement Key pair of the TPM
tpm_sealdata	Seal input data to the system TPM
tpm_selftest	Request TPM perform selftest and report
tpm_setactive	Change TPM active states
tpm_setclearable	Disable TPM clear operations
tpm_setenable	Change TPM enable states
tpm_setoperatorauth	Set the operator authorization value in the TPM
tpm_setownable	Change whether the TPM allows tpm_takeownership operations
tpm_setpresence	Change TPM physical presence states or settings
tpm_takeownership	Set up an owner on the TPM
tpm_version	Report TPM version and manufacturer information

Table 2.5. : Overview of the TPM Tool commands.

Table 2.5 lists the commands currently available from TPM Tools.
As an example for a command to set up an owner on a TPM, enter the following commands (as root):

```
# tpm_takeownership
```

Table 2.6 shows the five commands for PKCS#11 data management.

Command	Description
tpmtoken_import	Import an X.509 certificate and/or an RSA key pair into the user's TPM PKCS#11 data store
tpmtoken_init	Initialize the user's TPM PKCS#11 data store
tpmtoken_objects	Display the objects in the user's TPM PKCS#11 data store
tpmtoken_protect	Encrypt or decrypt data using a symmetric key stored in the user's TPM PKCS#11 data store
tpmtoken_setpasswd	Change the password(s) associated with the user's TPM PKCS#11 data store

Table 2.6. : PKCS#11 data management commands of the TPM Tools.

2.4.4. Sealing with TPM Tools

To illustrate the dynamic behavior between TPM tools and TSS, we describe as an example how an arbitrary file can be sealed to PCRs 12 and 14 using TPM Tools. Therefore, the `tpm_sealdata` utility is invoked with the following parameters:
`tpm_sealdata -i file.to.seal -o sealed.file -p 12 -p 14.`
The sequence diagram in figure 2.10 shows the flow of data and all relevant sub-functions. After invoking the tpm_sealdata function (step 1), `tpm_sealdata` retrieves random data from the TPM (step 1.7). To do this, the `tpmGetRandom` function invokes the method `Tspi_TPM_GetRandom()` of the class `TPM`. Then `tpm_sealdata` sets the SRK policy (steps 1.9 to 1.13) using the classes `Policy` and `Context`. The next functions (steps 1.15 to 1.21) build an RSA key object that will be created by the TPM. Then, an RSA key is created and loaded (steps 1.23 to 1.25). The subsequent functions build an encrypted data object that will hold the encrypted version of the symmetric key (steps 1.27 to 1.33). The final functions encrypt the given data and seal it to the symmetric key. It is possible to invoke this command with several command line parameters. For example, invoking `tpm_sealdata -p <PCR>` seals certain data to one or more PCR's. Upon success, the encrypted data is written to the file as specified with −o or printed out to stdout. Listing 2.9 shows the output of the successful sealing operation.

TrouSerS does not come with an unseal command line utility. In order to unseal data that has been sealed with `tpm_sealdata`, one has to use a function that comes within TrouSerS from the `tpmUnseal` family. If the conditions are right (i.e. the SRK has not changed and any PCR's specified at seal time are of the appropriate value), it will return the content of the file unsealed in a provided data buffer. A simple implementation would look like this:

```
if ((res=tpmUnsealfile(argv[1],&data,&size))==0){<do something>}.
```

Appendix A provides a simple implementation of an unseal utility called `tpm_unseal`.

Listing 2.9: Result of `tpm_sealdata`

```
-----BEGIN TSS-----
-----TSS KEY-----
AQEAAAARAAAABAEAAAABAAMAAQAAAAwAAAgAAAAAAgAAAAAAAAAAAAAABAM8d0Lh+
W79J99Al5KIXe25xIra0fd8yDMcXjmf/JeCK+c3V4oH5EnvdLfwVe/IbDfrnBEtQ
cn2OnweXNwIDpWNmjwbvrZhQCw1k4pIa45uGM/qo0Vfm/WbJ/JbXR2aXTcE+Xd/V
Oij+3XHHtddZjkPUtQUiHsj5pQA+gVkppNISMmrR/WbzIIUewC/GjH+DHjHm2+aW
Niu77ncvG6pbpFrPuxuJwYeqe2u8xsbXYdC5Vx4ou7+8JSq2nyoWDE0zL7lVrzBB
EWJWU+6dVX2fGMp1slK3V1XP4hzZmkI/1pcvLe06Pbmdc+Kk+u6Mi44Qqnwpn6eu
AImFxom3I8gg/nMAAAEAmyKLfMbJh5ZXLcif4dEltRt2kI+oI/3aS5D+SyFK74CO
1ohACDATJyQf0I+KmA52X9tG35CIwzQFBhAZsNf7LJ2RLNk25t6SWdV5BF3KWa2y
Y7kMk9qlgyEK852nwQ3hwKf6kdJJRc6cq1jUjqlBCFHFZ+/pWYsrskxenA0WssV8
fG3MeXZxau+5CfDTGoTrpTCJmzcxtQSCQu7puzcLvo4mxyuldIi1ImcXGJwBBHgA
L9AAVeDMqeajz9YuiFqV0AuqgqVQrKSiAc0c0tUvtlJ/YeCtzvlUf8HrFZrGmWlp
rsu2Q5u/OxbXVkPrZG7HiFqz90hhrsPA78dD7The4A==
-----ENC KEY-----
Symmetric Key : AES-256-CBC
AQEAAAAAACwAAgBQhBbi5yRMhzTZsLVa4Xk68CSuR1+EFuLnJEyHNNmwtVrheTrw
JK5HXwAAAQBRE91fMOjAS5U0RD+K3EA9aZQ2cLw3J4BO+65mjKo4Bm8KJgw6rFeq
TbuXl0qCdOq/env0Eb5V3+/sx3wOzBN6bIVUEbvk9DxcXA9LiJy/X4vR2wdVtaEJ
N49+MWTeEma95QV/gzvM4bUNEsiQdxg+fzbgGG6aoOjDbND1/ATBXCRxYJ+zxky9
O/ RrPrZZV6qsV1mAr2ERMdxs1AHp5CAMMfRWQdwC0hmD0RkciqPW7WE6xdhxp2rp
L8NAUgeVDwe2m90a4FhQFYflRB0vknfn1O5t/JtsGLD1lQQ8DZQVZhhEyfiRRbgE
wENOfacFeteAE+YcxPfYzOpCLIIoSwFM
-----ENC DAT-----
vWe+mpv7TVAsXs89t68eeA==
-----END TSS-----
```

2.4.5. Quality of the Implementation

Code complexity. IBM has written its implementation in ANSI C and the code consists of about 5400 lines. The code is partially documented but well structured. The most functions are clear and brief, usually not over 50 lines and less than four parameters. The largest part is the library; the folders including sources for data and TPM management are roughly the same size. Table 2.7 shows the exact allocation of data.

Structure and style. The code is well-structured. Most functions are clear and brief, usually consisting of less than fifty lines and four parameters. The exception handling is implemented with goto statements.

Figure 2.10.: Data flow after invocation of tpm_sealdata.

High-level design. All commands are to be executed from the command line. They make use of the same library.

Documentation and support. The code is partially documented. For every command there exists a man page which is also available under http://trousers.sourceforge.net/ man.html. Requests for support can be given to http://trousers.sourceforge.net/.

Limitations. The latest release of TPM Tools contains support for the `TSS_WELL_KNOWN_SECRET` of the Storage Root Key (SRK) as used by TrouSerS. This does not hold for `tpm_sealdata` and `tpmUnsealfile`, therefore the source code had to be patched in order to perform the interoperability tests in section 3.1. A bug report has been sent to the maintainers. The used patches are printed in Appendix B.

Directory	Lines of code
lib	1944
src_data_mgmt	1520
src_tpm_mgmt	2089
include	242
src_cmds	231

Table 2.7. : Code complexity of the TPM Tools.

2.5. TPM Manager

The goal of the "TPM Manager" project [14, 21] is the realization of open source TPM management software providing an easy-to-use graphical user interface. Currently, the TPM Manager can be used under Linux, while later releases should also be usable as an isolated application executed on top of a security kernel such as Turaya[15] or OpenTC[16] . Current releases of the TPM Manager were developed by Sirrix AG[17] and Ruhr-University Bochum[18] . The TPM Manager is published under version 2 of the GNU GPL license.

2.5.1. Installation and Configuration

The TPM Manager is hosted on sourceforge at http://sourceforge.net/projects/tpmmanager/. The latest source snapshot, Version 0.4 at the time of writing, is also available. The distribution is based on `automake/autoconf`; therefore, it can be configured and installed as described in Listing 2.10.

15 http://www.emscb.com/
16 http://www.opentc.net/
17 http://www.sirrix.com/
18 http://www.trust.rub.de/

Listing 2.10: Configuring and compiling the TPM Manager

```
# tar xzf tpmmanager-0.4.tar.gz
# cd tpmmanager-0.4
# ./configure
# make
# make install
```

Since the TPM Manager is based on some KDE widgets, corresponding header and library files of KDE and Qt should be in the library path. The TPM Manager has been successfully compiled under KDE 3.5 and Qt3.

2.5.2. High-Level Design

As illustrated in figure 2.11, the design of the TPM Manager has two layers, the Graphical User Interface (GUI) layer and the MicroTSS layer.
While the GUI layer provides the user interface components, based on widgets of the KDE[19] and QT[20] frameworks, the MicroTSS layer offers a simple, object-oriented interface to the functions of the underlying TPM.
Although the current implementation of the MicroTSS is based on the TrouSerS TSS (see Section 2.3), the abstraction provided by the MicroTSS layer allows for support from alternative TSS implementations offered by a third party or from the required functions implemented directly. The latter is especially important if the TPM Manager should be used in a security-critical environment, e.g., as a part of a security kernel.

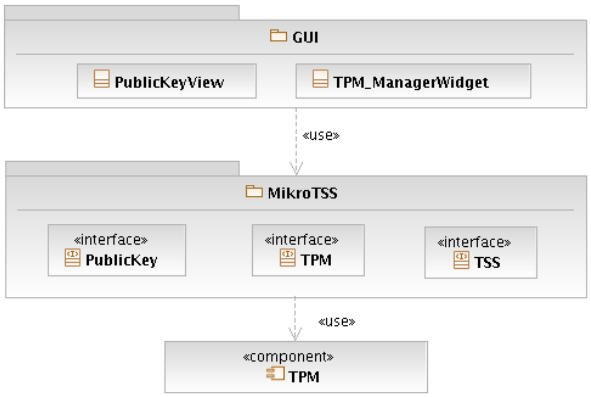


Figure 2.11.: The high-level design of the TPM Manager.

19 http://www.kde.org/
20 http://www.troll.no/

The GUI layer. The GUI layer of the TPM Manager offers a graphical interface and uses the MicroTSS layer to access TPM functions. The logic of the user interface layer is realized by the Qt-based class, `TPM_ManagerWidget`.
Its base class, `TPM_ManagerWidgetBase`, is created by the Qt Designer, Trolltech's tool for designing and building graphical user interfaces (GUI's). The Qt Designer allows the addition of new functionality without much effort. Figure 2.12 shows an example screenshot of the TPM Manager GUI in use.

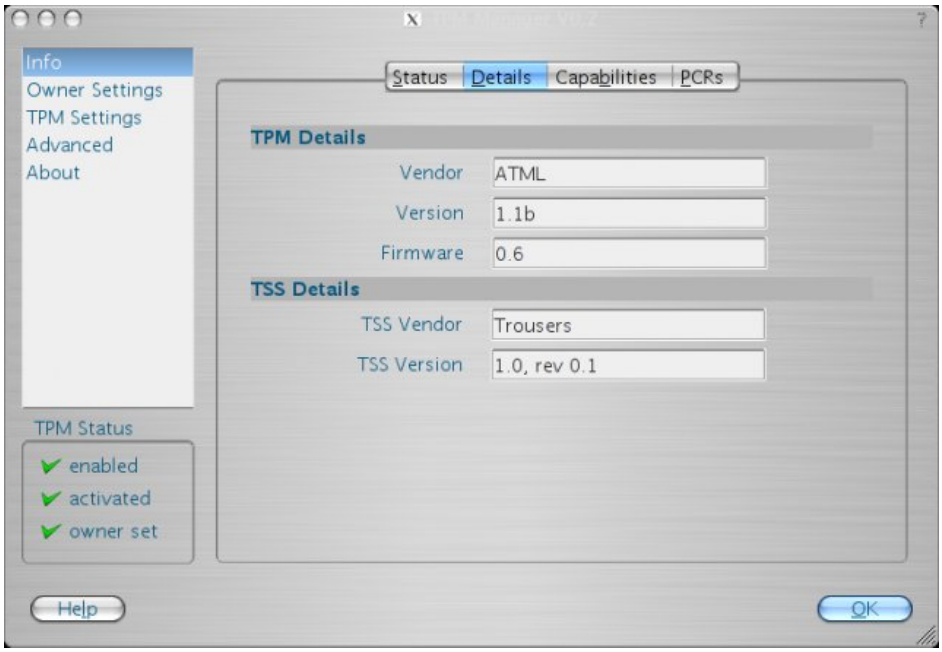


Figure 2.12.: Example Screenshot of the TPM Manager.

The MicroTSS layer. The MicroTSS layer provides an abstract interface to access the TPM and hides implementation details. Figure 2.13 shows the Unified Modeling Language (UML) model of the public interfaces and components provided by the TPM Manager and the TSS Service Provider Interface. The main component is the class `TPM` that implements the TPM management functions. The class `TPM` is created by the class `TSS` managing access to the TSS implementation in use. For example, the class `TSS` checks the availability of a TPM driver before creating an object of type TPM. The class `PublicKey` provides information about types and attributes of cryptographic encryption and test keys managed by the TSS.
Although the MicroTSS interface includes a small number of functions related to version 1.2 of the TSS specification, the main portion is based on TrouSerS TSS version 1.1b.
The current implementation of the class `TPM` itself uses the TSPI interface provided by the `libtspi.so` library of TrouSerS. Since the TSPI interface includes many functions, in figure 2.13, only the functions that are required to realize the use case "Take Ownership" are described in Section 2.5.3.

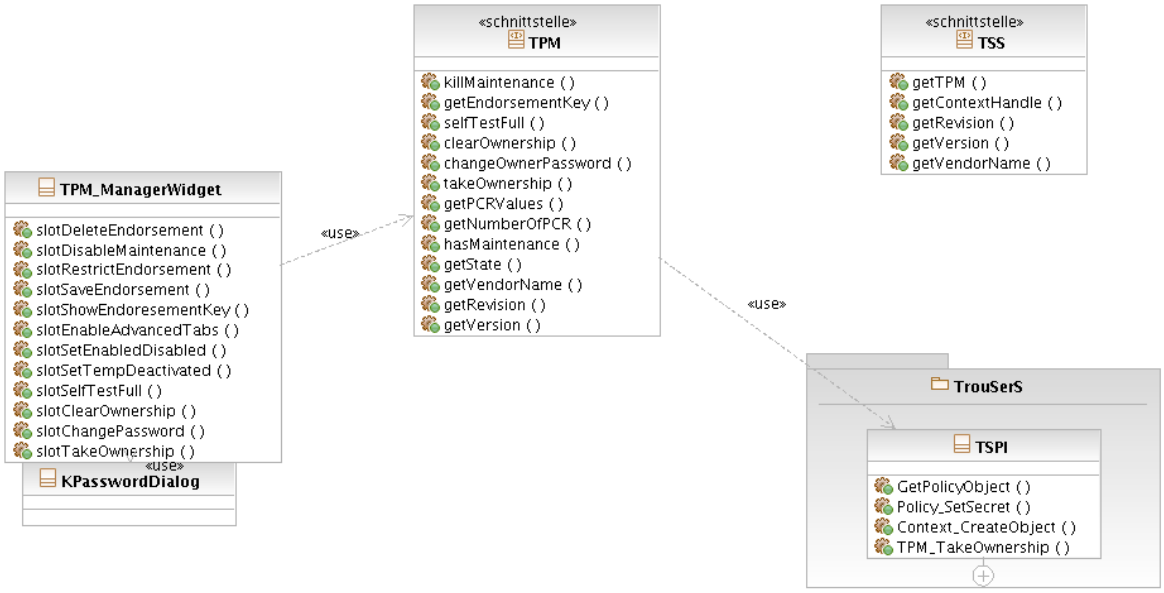


Figure 2.13.: Interfaces provided and used by the TPM Manager and the MicroTSS.

2.5.3. Taking Ownership with the TPM Manager

To illustrate the dynamic behavior of the TPM Manager design, the use case “Take Ownership” is given as an example. The sequence diagram in figure 2.14 illustrates the message and control flow upon invocation of the “Take Ownership” function in the TPM Manager. After the user presses the appropriate button of the TPM Manager dialog, the method `slotTakeOwnership()` of the class `TPM_ManagerWidget` is invoked (step 1). Here, two password dialogs are created for entering the TPM owner password and the SRK password (steps 1.1 and 1.3). Next, the MicroTSS, precisely the method `takeOwnership()` of the class `TPM`, is invoked (step 1.5). This method hides the complexity of the interface of the `libtspi.so` by invoking TSPI functions setting the owner password and the SRK password to finally take ownership of the TPM (steps 1.5.1 to 1.5.11). finally, a dialog informs the user about the result of the take ownership operation (step 1.7).

2.5.4. Quality of the Implementation

Code complexity and design. The TPM Manager is implemented in C++ based on a UML model created and maintained using Rational Software Architect developed by IBM. The GUI package of the TPM Manager currently includes around 3200 source lines of code, while the MicroTSS package includes only 857 Source Lines Of Code (SLOC).

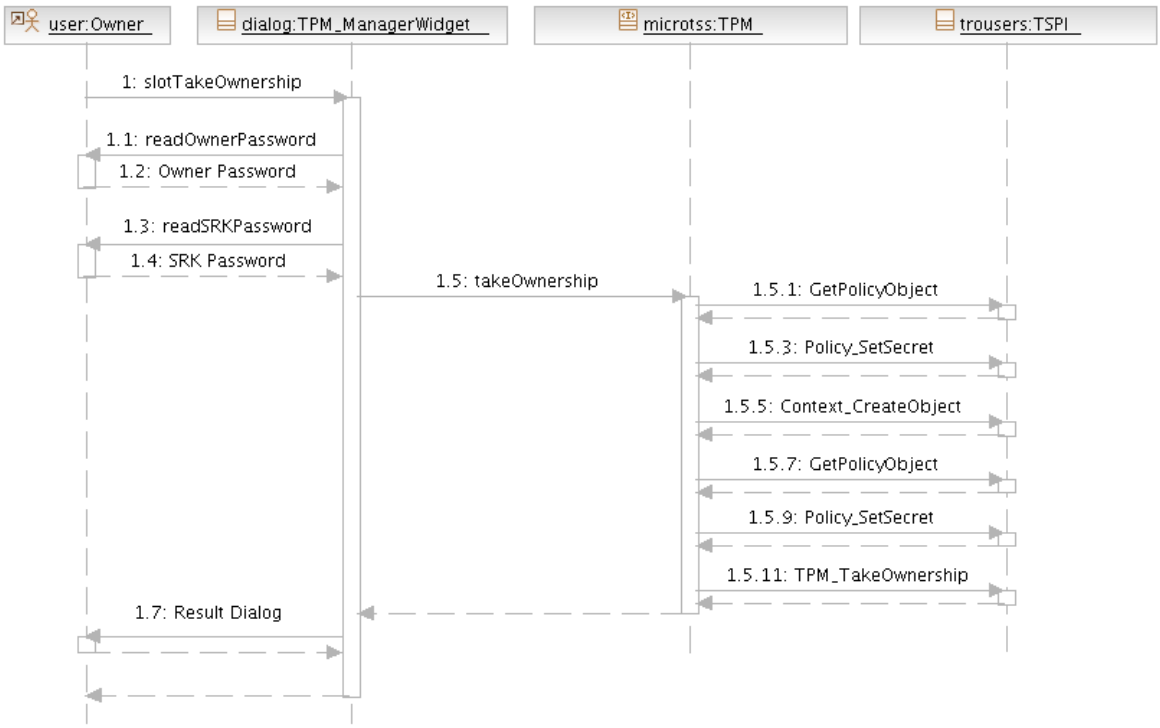


Figure 2.14.: Control flow after invocation of the "Take Ownership" function in the TPM Manager.

Documentation and support. The technical report [21], included in the TPM Manager source distribution, contains detailed descriptions of the realized use cases and the high-level design. Implementation details and a user manual are currently missing. Mailing lists, a bug tracker and a discussion forum are currently hosted on sourceforge.

Limitations. Currently, the most important TPM management functions are supported. However, some limitations have already been identified:

- Creation and deletion of the Endorsement Key is currently not supported.

2.5.5. Future Work

The following extensions of the TPM Manager are already in progress and scheduled for the next minor release:

- fixing of minor bugs.

The following tasks are planned for the long term:

- Porting to TSS version 1.2 and implementation of use cases not currently implemented.
- Porting to another widget library of less complexity, e.g., the Fast Light Toolkit[21] (flTK).

2.6. OpenSSL TPM Engine

OpenSSL [13] is an open source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols, including the necessary cryptographic operations. The toolkit is based on the SSLeay library [28] developed by Eric A. Young and Tim J. Hudson. It is available under an Apache-style license, meaning that it can be used for both commercial and non-commercial purposes.
The OpenSSL library supports the creation, manipulation and usage of cryptographic modules in the form of engine objects, i.e., containers for implementations of cryptographic algorithms. Since these objects can be loaded dynamically, OpenSSL can be extended by alternative software implementations or hardware modules such as smartcards or the TPM. The cryptographic functionality that can be provided by an OpenSSL engine includes the following abstractions:

- Alternative RSA, (EC)DSA, and (EC)DH implementations
- Symmetric cipher algorithms
- Hash algorithms
- Key loading
- Storage of cryptographic keys and certificates

2.6.1. OpenSSL Engine Interface

All functions required to manage and implement OpenSSL engines are defined in the file openssl/engine.h. Figure 2.15 shows the functions used by an OpenSSL engine to add support for new cryptographic primitives. Implementations of the OpenSSL engine interface can optionally define a subset of those functions. In this case, the default implementations of the other operations are used.

21 http://www.fltk.org/

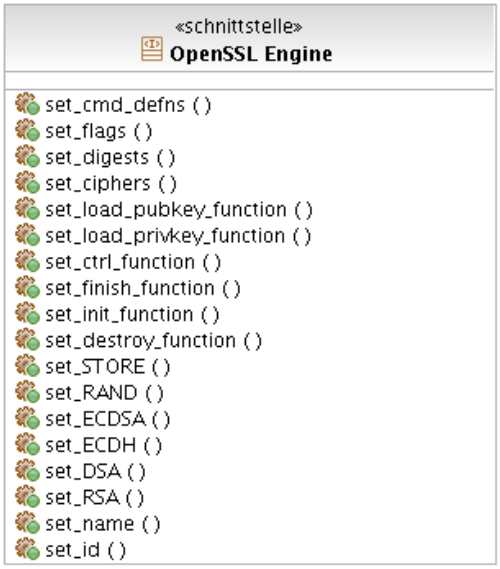


Figure 2.15.: The generic OpenSSL engine interface.

2.6.2. OpenSSL Engine Activation

Unfortunately, the current design and implementation of the OpenSSL engine support is not transparent to the application layer. Instead, applications must invoke OpenSSL functions to explicitly initialize and register available OpenSSL engines. Two alternative approaches are possible for this: Either the appropriate engine is activated in the source code of the application directly, as shown in Listing 2.11, or the application has to activate OpenSSL configuration support by defining `#define OPENSSL_LOAD_CONF`.

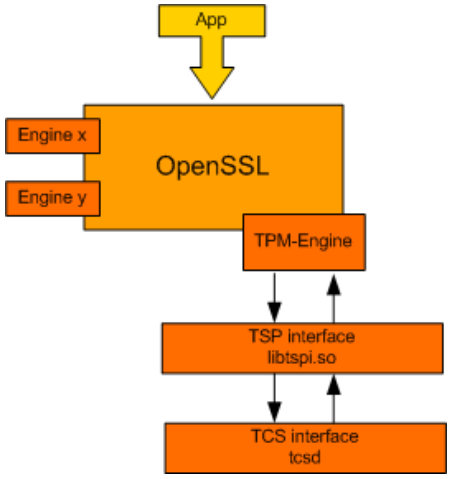


Figure 2.16.: Interaction between OpenSSL and TSS.

2.6.3. The TPM Engine

The OpenSSL TPM engine, which is part of the TrouSerS project [9], implements the OpenSSL engine API to allow cryptographic operations to be performed by a TPM. In the following, we discuss version 0.4.1 of the OpenSSL engine.
As illustrated in figure 2.16, the TPM engine connects the TPM to OpenSSL via the TSS. More concretely, to communicate with the TPM, the OpenSSL TPM engine itself uses the shared library `libtspi.so` that is part of the TSS.

Listing 2.11: Invocation of an OpenSSL engine

```
/* Make the "dynamic" ENGINE available */
void ENGINE_load_dynamic ( void ) ;
/* Make the CryptoSwift hardware acceleration support available */
void ENGINE_load_cswift ( void ) ;
/* Make support for nCipher's "CHIL" hardware available */
void ENGINE_load_chil ( void ) ;
...
/* Make ALL ENGINE implementations bundled with
OpenSSL available */
void ENGINE_load_builtin_engines ( void ) ;
```

2.6.3.1. Configuration and Installation

The OpenSSL TPM engine distribution openssl_tpm_engine−0.x.y.tar.gz can be downloaded from the TrouSerS website at http://trousers.sourceforge.net/. Since `automake/autoconf` is supported, it can be configured and installed using the commands given in Listing 2.12.
Ensure that the path to the existing OpenSSL installation is correct, or else OpenSSL will not be able to find the TPM engine. To use the TPM engine, a code fragment such as that defined in Listing 2.13 has to be inserted:

Listing 2.12: Configuration and installation of the OpenSSL TPM engine

```
# tar xzf openssl_tpm_engine-0.x.y.tar.gz
# cd openssl_tpm_engine-0.x.y
# ./configure --with-openssl=/path/to/openssl
# make
as root
# make install
```

Listing 2.13: Invocation of the OpenSSL TPM engine

```
/* Make all engine implementations bundled with
OpenSSL available */

void ENGINE_load_builtin_engines ( void ) ;

/* Select TPM engine */
ENGINE *e = ENGINE_by_id ( "tpm" ) ;

/* Initialize TPM engine */
ENGINE_init ( e ) ;

/* Make t h e TPM engine the default */
ENGINE_set_default_RSA ( e ) ;
ENGINE_set_default_RAND ( e ) ;
```

2.6.3.2. Quality of the Implementation

Code complexity. The OpenSSL TPM engine is implemented in C and consists of about 1825 SLOC. The provided test suite consists of 174 SLOC.

Tests. The OpenSSL TPM engine comes with a simple test suite including four tests:

1. Load a TPM key from file using the engine load command.
2. Do a test run on the loaded TPM key.
3. Call stir random through the RAND interface.
4. Test auth data passthrough to the engine.

Documentation and support. The source code of the TPM engine is not extensively documented. Only a short README and OpenSSL configuration examples are provided covering two use cases. Support is provided via a mailing list in the TrouSerS project.

Problems and missing features. Unfortunately, appropriate documentation on how to use OpenSSL engines is missing and there has been no response to queries from the OpenSSL mailing list. Thus, it was not possible to use the TPM engine from another program using the provided OpenSSL configuration file.

2.6.4. TPM-based Transport Layer Security with the OpenSSL TPM Engine

To illustrate the dynamic behavior of the OpenSSL TPM engine, this section describes how a TPM-based transport layer can be created. Since the TPM offers the generation of legacy RSA keys, this section describes the process necessary to use a TPM RSA key within OpenSSL in order to achieve transport layer security. Additionally, this section shows how to generate a certificate of the TPM key for verification purposes and usage in a web server. The required steps to set up an encrypted channel using a TPM-shielded key are:

1. Install the OpenSSL TPM engine as described in Section 2.6.3
2. Create signature keys:
   ```
   create_tpm_key <keyfilename>
   ```
3. Create a self-signed certificate:
   ```
   openssl req -keyform engine -engine tpm -key <keyname> -new \
   -x509 -days 365 -out <certfilename>
   ```
4. Start an OpenSSL test server:
   ```
   openssl s_server -cert <certname> -www -accept 4433 \
   -keyform  engine -engine tpm -key <keyname>
   ```

Figure 2.17 illustrates the control flow of Step 1 and Step 4. To test the result, open a web browser pointing to the appropriate port, e.g., by entering konqueror https://localhost:4433. The browser should show an OpenSSL information page using an encrypted communication channel.

2.6.5. Future Developments

According to the OpenSSL documentation, the engine API and internal architecture is currently under review. Slated for possible release in v0.9.8 is support for transparent loading of “dynamic” engines (built as self-contained shared libraries). This would allow engine implementations to be provided independently of OpenSSL libraries and/or OpenSSL-based applications, as well as remove any requirements for applications to explicitly use the “dynamic” engine to bind to shared library implementations.

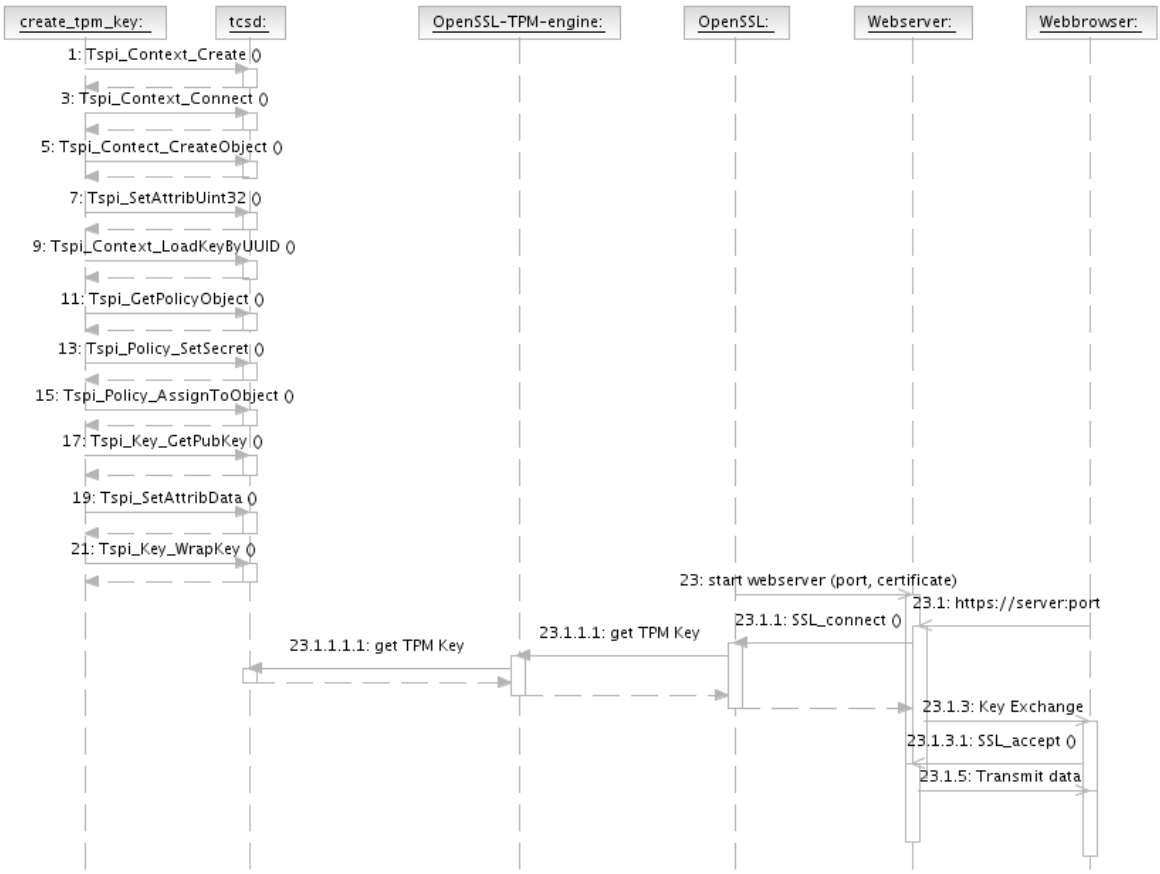


Figure 2.17.: General (informal) sequence of TPM engine with OpenSSL.

3. Compliance and Interoperability

The goal of this chapter is to check the compliance and the interoperability of the components analyzed in Chapter 2. To test the interoperability of these components, several test cases have been evaluated based on a variety of open source operating system architectures including SE-Linux, the Xen hypervisor, and the Turaya security kernel.
Compliance analysis (Section 3.1) is conducted, where possible, to determine the extent to which open source components are compliant to existing specifications. The test environments of the interoperability tests are explained in Section 3.2, while the test results are discussed in Section 3.3.

3.1. Compliance Analysis

3.1.1. Bootloader Compliance

The compliance analysis of the introduced bootloaders depends on two different measurement approaches. TrustedGRUB and GRUB-IMA make usage of the Static Root of Trust for Measurement, while OSLO uses the Dynamic Root of Trust for Measurementfor its measurements. While the needed functionality for the Static Root of Trust for Measurement is provided by the BIOS, the Dynamic Root of Trust for Measurementuses vendor specific instructions. In both approaches, the measurements are stored inside the TPMs PCRs, which are defined in the TCG PC client specifications [22, 26] and shown in Table 3.1 and Table 3.2.

3.1.1.1. TrustedGRUB

As described in Section 2.1.1, TrustedGRUB extends PCRs with measurements of the multiboot modules to be loaded. To do so, it relies on appropriate BIOS support of the TCG-enabled platform. TrustedGRUB uses the PCRs defined in Table 3.3:
Since the TCG PC client specification defines the usage of PCRs 0-7, but does not explicitly define the usage of the upper registers 8-15, the current TrustedGRUB implementation complies with the TCG PC client specification [22, 26].
To verify the measurements performed by TrustedGRUB and pre-calculate measurement values, the TrustedGRUB distribution provides the utility `verify_pcr`. For example, to verify the resulting PCR value after measurement of the file `/boot/vmlinuz`, the following command can be executed within a shell:

```
$ verify_pcr NULL /boot/vmlinuz
-> Result for PCR:
39 21 82 94 e8 d5 0a b4 2e 3e 84 64 cc 41 3c 40 ab ca b5 0a
```

PCR Index	Alias	pcrReset
0 - 15	**Static RTM**	**0**
0	CRTM, BIOS and Platform Extensions	0
1	Platform Configuration	0
2	Option ROM Code	0
3	Option ROM Configuration and Data	0
4	IPL Code	0
5	IPL Code Configuration and Data	0
6	State Transition and Wake Events	0
7	Reserved for future usage	0
8 - 15	unspecified	0

Table 3.1. : S-RTM-PCR usage as defined by the TCG.

PCR Index	Alias	pcrReset
16 - 23	**Dynamic RTM**	**1**
16	Debug	1
17	Locality 4	1
18	Locality 3	1
19	Locality 2	1
20	Locality 1	1
21	T/OS Controlled	1
22	T/OS Controlled	1
23	Application Specific	1

Table 3.2. : D-RTM-PCR usage as defined by the TCG.

The `verify_pcr` utility also works with the content of PCR 13, containing all arbitrary files loaded via the checkfile method. As described earlier, the check file contains a list of files to be measured, while the result is used to extend a PCR.

```
$ verify_pcr NULL /etc/passwd /etc/shadow [...]
-> Result for PCR:
6b c9 23 12 c7 32 36 90 ff 7d 5a d0 e0 d2 a6 00 29 9b 10 b0
```

This shows that (i) the measurements have been correctly calculated and (ii) the measurements have been correctly extended into the PCR.

PCR	Usage
4	MBR information and `stage1`
8	`stage2` Part 1
9	`stage2` Part 2
12	Command line arguments
13	`checkfile` measurements
14	Kernel and modules

Table 3.3. : PCR usage of TrustedGRUB.

3.1.1.2. GRUB-IMA

The GRUB-IMA measurements are stored inside the PCRs defined in Table 3.4. The PCR usages complies with the TCG PC client specification [22, 26]. In order to verify the measurements, one can observe the ACPI PCR extension log under Linux:

```
# mount -t securityfs /sys/kernel/security
# cat /sys/kernel/security/tpm0/ascii_bios_measurements
# cat /sys/kernel/security/tpm0/binary_bios_measurements
```

PCR	Usage
4	MBR information, `stage1` and `stage2`
5	ACTIONs (here: EventLog)
8	OS components (kernel, initrd, module, `measure` command)

Table 3.4. : PCR usage of GRUB-IMA.

The `binary_bios_measurements` contains the exact BIOS ACPI table (which itself includes the Integrity Measurement Log (IML)). The `ascii_bios_measurements` is the human-readable format of this. Listing 3.1 shows an example of the BIOS log.
In this example:

Listing 3.1: Example output of `ascii_bios_measurements`

```
0 98a1f3605524e874d068ac05be199cb0cb6c5e04 08 [S-CRTM Version]
0 ae81f0c566a34dbcefe204ade2b863622355daf4 01 [POST CODE]
0 d62689661aa4d0c5c93f4ed92b8fa7d592a45cd3 01 [POST CODE]
0 dd261ca7511a7daf9e16cb572318e8e5fbd22963 01 [POST CODE]
<snip>
4 c1e25c3f6b0dc78d57296aa2870ca6f782ccf80f 05 [Calling INT 19h]
4 38f30a0a967fcf2bfee1e3b2971de540115048c8 05 [Returned INT 19h]
4 cfa550a3b0fa6f8be76b8cf2d68be28355e57e2f 05 [Booting BCV Device 80h]
4 d6cf7a4dc273b2750b9791a2ba760c6fc6dc14a4 0d [IPL ]
5 1c47735c2cd4a0e2f33ca1730f85038f18d45e30 0e [IPL Partition Data ]
4 f3e37129afac034af2edba9d46f3571b5113ca0a 0d [IPL]
4 2929efe0b6d3582c0b23ea287b278b85d5cf33ee 0d [IPL]
4 5dbe0ae807ae0ecdb3e489603457d19c445c9c46 0d [IPL]
4 f6019d76ba7d20659d28d225a3305c9485533f95 06 []
4 8cdc27ec545eda33fbba1e8b8dae4da5c7206972 04 [GRUB Event Separator]
5 8cdc27ec545eda33fbba1e8b8dae4da5c7206972 04 [GRUB Event Separator]
5 cb9384a8c98849f08e21da4842fac3fd7cd9a1a0 0e [IPL Partition Data]
5 e32ba4121c9624586564659738014 33b537ec85e 05 [Password Prot. using MD5]
5 d63d12ced978aca120bfe6ee7683e394c2ffaef0 05 [Boot Seq. User Interv.]
5 486a8ca8bb98a87b3c0e5cb0e375b4640bad6bb9 1105 []
8 0b2adebe5ff3ee1f1f8a77e3cdba7f82948516ab 1205 []
8 1287bdda3d76ebed9b705d51c6f03b8f06cbf5a1 1305 []
5 2431ed60130faeaf3a045f21963f71cacd46a029 04 [OS Event Separator]
8 2431ed60130faeaf3a045f21963f71cacd46a029 04 [OS Event Separator]
8 fac33a1fc0ad42c07d00322d64c23f67567f334a 1005 []
```

```
8 0b2adebe5ff3ee1f1f8a77e3cdba7f82948516ab 1205 [] is linux kernel
8 1287bdda3d76ebed9b705d51c6f03b8f06cbf5a1 1305 [] is initrd image
```

Thus, it is possible to validate this measurement by using sha1sum.

```
# sha1sum /boot/vmlinuz-2.6.18-8ima.el5
0b2adebe5ff3ee1f1f8a77e3cdba7f82948516ab /boot/vmlinuz-2.6.18-8ima.el5

# sha1sum /boot/initrd-2.6.18-8ima.el5.img
1287bdda3d76ebed9b705d51c6f03b8f06cbf5a1 /boot/initrd-2.6.18-8ima.el5.img
```

3.1.1.3. OSLO

The OSLO measurements are stored into the PCRs defined in Table 3.5.

PCR	Usage
17	Measurements
19	Command line hashes

Table 3.5. : PCR usage of OSLO.

OSLO - as a security enhanced bootloader - is allowed to use localities 2 and 3, since those localities are designed for trusted operating systems and trusted initialization software. The PCR usage for locality 2 is PCR 19, therefore OSLO's PCR choice for the command line hashes is compliant. For OSLO's binary measurements PCR 17 is used, but PCR 17 is associated with locality 4 and should therefore only be used by special initialization hardware.

3.1.2. TrouSerS TSS

Compliance tests. IBM considers their TSS 1.1 implementation to be complete. However, an analysis of version 1.1 of TrouSerS is not part of this study.
Regarding the progress of the TSS 1.2 implementation, the maintainers provide detailed information about the current state on their project homepage at http://trousers.sf.net/trousers12support.html. Since TrouSerS 1.2 is still under heavy development, it is impossible to draw conclusions at the moment regarding its compliance with the TSS implementation. According to the TrouSerS website, IBM is currently working on 42 TSS-related services, which can be categorized into four states:

1. Complete feature sets: 19
2. Incomplete feature sets: 3
3. In progress: 5
4. Implementation not started: 15

Non-compliance. According to the TrouSerS project page, the implementation explicitly breaks the TSS specification at four points:

1. By default, policies are created with their secret mode set to TSS_SECRET_MODE_NONE as opposed to TSS_SECRET_MODE_POPUP. In a GUI environment where the application fails to set policy secrets where it should, a TSS_E_POLICY_NO_SECRET response is returned instead of a popup window asking for a secret. In a non-GUI environment where the application does not set policy secrets where it should, TSS_E_POLICY_NO_SECRET is returned when the TSS tries to launch its popup, as opposed to TSS_E_INTERNAL_ERROR. The application is, of course, still free to manually set its policies to TSS_SECRET_MODE_POPUP.

2. In Tspi_TPM_SetStatus, physical presence commands can be passed down to the TCS by default. The TCS will check what run level the machine is in. If it is in single-user mode, execution of the physical presence command is allowed. This enables hardware without BIOS support for the TPM to enable, disable and clear the TPM. These commands have no effecton hardware with BIOS support.
3. The TrouSerS v1.1 TSS (versions 0.2.5+) supports callbacks in the TSS v1.2 style. According to the TSS v1.1b specification, callbacks are set using a constant-width, 32-bit field in Tspi SetAttribUint32. This restricts 64-bit applications from being able to set callbacks. To enable 64-bit TSS v1.1 applications, TrouSerS offers support for TSS 1.2 style callbacks by default.
4. The TrouSerS v1.1 TSS (versions 0.2.5+) supports the TSS v1.2 attribute for control ling the NULL-terminating data of passwords.

Although TrouSerS breaks the TSS specification as described above, it is possible to enable strict specification compliance by compiling it with the following command line argument:

```
./configure --enable-strict-spec-compliance
```

3.1.3. OpenSSL TPM Engine

The OpenSSL TPM Engine currently implements the following cryptographic operations:

- RSA public key encryption, RSA private key decryption, RSA signature, RSA verification and RSA key generation using the TPM
- Random number generation using the TPM
- Loading of public RSA keys and private RSA keys
- Three configuration commands

The three supported configuration commands are (i) TPM_CMD_SO_PATH, to define the path to the libtspi.so shared library; (ii) TPM_CMD_PIN, to set the SRK secret; and (iii) TPM_CMD_SECRET_MODE to select the TSS secret mode used for all secrets.
Due to limitations of the cryptographic hardware within a TPM, other asymmetric and symmetric ciphers cannot be supported by TPMs. However, neither the use of the TPM-internal SHA-1 engine (using `ENGINE_DIGESTS_PTR`) nor the TPM-internal non-volatile storage (using `STORE_METHOD`) is currently supported.

3.1.4. TPM Manager

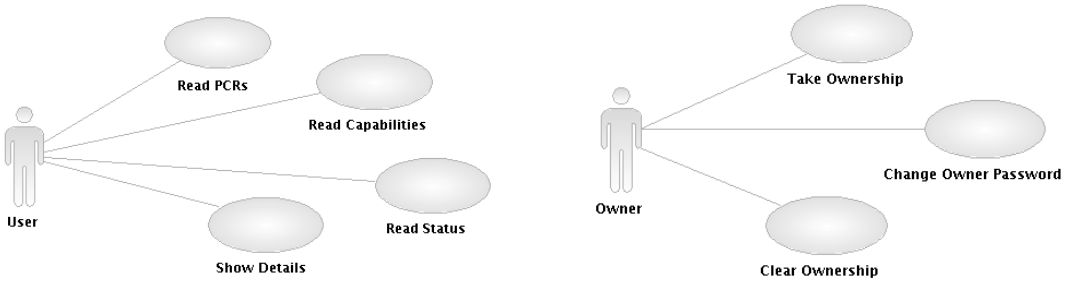
The latest stable version of the TPM Manager at the time of writing is version 0.4, which is based on TrouSerS version 0.2.x implementing TSS version 1.1b. Although it is possible to compile and use version 0.4 of the TPM Manager with TrouSerS version 3.x, it is not recommended. This is because many functions do not work correctly due to changes in the behavior of the TrouSerS implementation. To test against a version 1.2 TPM, the latest development version, 0.5, of the TPM Manager and TrouSerS version 0.3.1 was used.
The functionality provided by the TPM Manager can be split into the following functional packages:

- TPM Info: Generic information about the current TPM state
- Owner Settings: Owner-related TPM operations
- TPM Settings: TPM-specific operations
- Maintenance: Backup and restoration of the TPM state
- EK Management: Endorsement key management operations

The following paragraph discusses the TPM management operations provided by these functional packages in more detail.

TPM Info. This functional package provides general information about the TPM state, i.e., the TSS and TPM driver found, the availability and type of the Endorsement Key, the TPM capabilities, and the current PCR status (see figure 3.1). The TPM driver status is currently hard-coded, since the TSS specification does not provide information about the availability of the TPM driver. Therefore, the TPM Manager currently assumes that a valid TPM driver exists if a TCSD daemon is found. Moreover, version 0.4 of the TPM Manager shows the number of available PCRs only as capabilities.

Owner settings. Here, the user/owner can take ownership of the TPM, change the owner password and clear an existing TPM ownership (see figure 3.1). A missing feature of version 0.3 is the possibility to use the TCG_WELL_KNOWN_SECRET as an SRK password when taking ownership, because this is important to guarantee compatibility with parallel Windows installations. This feature has been supported since TPM Manager version 0.4.

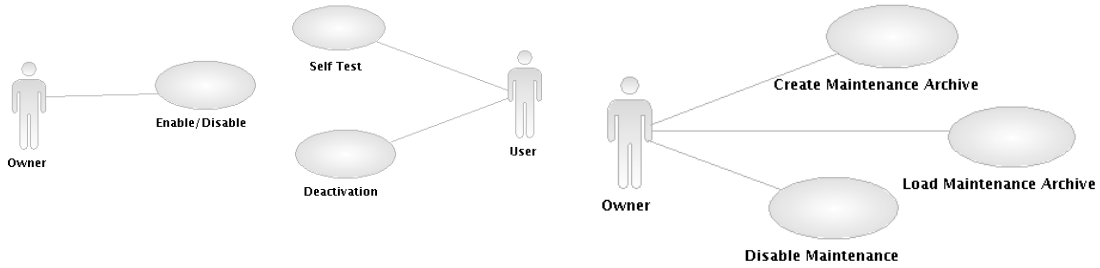


(a) TPM Info (b) TPM Owner

Figure 3.1.: Use cases realized by the functional packages TPM Info (a) and TPM Owner (b).

TPM settings. This functional package allows the user to change some TPM specific settings, e.g., to enable/disable or deactivate the TPM, to invoke the TPM-internal test functions and to update the TPM firmware (see figure 3.2). Although some TPM models (e.g. some of the TPMs from Infineon) allow firmware updates, this functionality is currently not implemented due to the fact that the TPM specification lacks a detailed description.

Maintenance. TPM Maintenance includes the possibility to create maintenance archives (i.e. backups of the SRK), load them, and permanently disable the maintenance functionality (see figure 3.2). Maintenance is, however, an optional feature of the TCG specification and, to our knowledge, not implemented by the TPMs currently available. Therefore, this functionality has not been implemented to this point, as it can neither be used nor tested.



(a) TPM Settings (b) TPM Maintenance

Figure 3.2.: Use cases realized by the functional packages TPM Settings (a) and Maintenance (b).

EK management. This functional package provides functions to manage the Endorsement Key (EK) of the TPM. Such features as creation, deletion and reading the EK or EK certificate as well as preventing the reading of the EK without owner authorization (see figure 3.3). Creation and deletion of the Endorsement Key is currently not implemented.

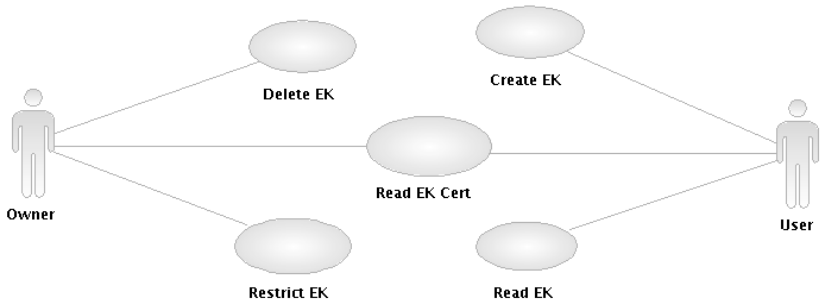


Figure 3.3.: Use cases realized by the functional package EK Management.

3.2. Test Environment and Test Cases

In the following, we describe the test environment and test cases used to evaluate the interoperability of the security architectures discussed in the following subsections, which use the Trusted Computing components described in Chapter 2.

3.2.1. Test Environment

All interoperability tests were performed on an HP Compaq 6715b notebook with an Infineon version 1.2 TPM, 2 GB of RAM, and the latest official TrouSerS release (version 0.3.1). Moreover, the following three operating system architectures formed the basis of the interoperability tests.

3.2.1.1. Security-Enhanced Linux (SE-Linux)

Security-Enhanced Linux (SE-Linux) is an enhancement for Linux that can enforce a variety of security policies through the use of Linux Security Modules (LSM) [19, 18]. SE-Linux was developed primarily by the National Security Agency (NSA). Other organizatons, including RedHat, the Department of Defense (DoD), IBM, HP and Tresys, contributed as well. SE-Linux is comprised of a kernel module, patches to various security-related applications and a security policy.
In a standard Linux security model (with only discretionary access control), every program is allowed to control access to its resources. In contrast, SE-Linux allows a more fine-grained access control than standard Linux permissions offered by LSMs to enforce a Mandatory Access Control (MAC) security policy. This is realized by the principle of least

privileges. For example, a process in SE-Linux can be granted only the permissions it needs to be functional.

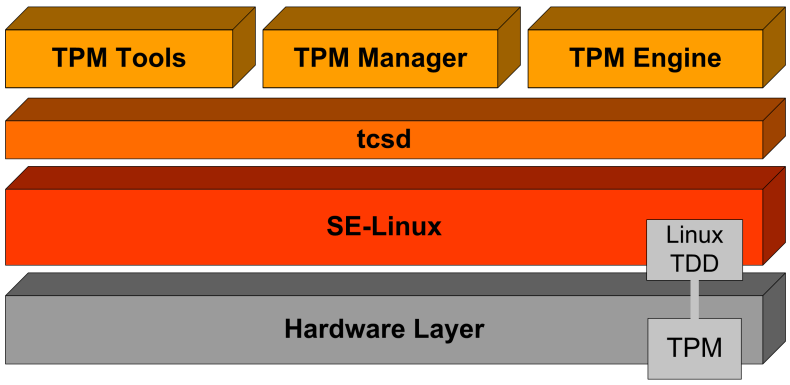


Figure 3.4.: SE-Linux-based architecture.

SE-Linux is based on the FLASK security architecture [20] originally developed to work through some of the inherent problems with a MAC enforcement policy. FLASK provides general support for enforcing many kinds of mandatory access control policies, including those based on the concepts of type enforcement, role-based access control and multi-level security.
The first SE-Linux release was made available in December 2000, based on Linux kernel version 2.2.12 and Red Hat version 6.1 utilities. As of this writing, SE-Linux has been integrated into many GNU/Linux distributions and can be downloaded from http://www.nsa.gov/selinux/. SE-Linux includes general-purpose security policy configurations designed to meet a number of security objectives and demonstrate the configuration of the security policy. The flexibility of the system allows the policy to be modified and extended to customize the security policy as required for any given installation.
The Trusted Computing architecture based on SE-Linux is illustrated in figure 3.4. The test environment is based on a SE-Linux-enhanced Gentoo[22] 2007.0-hardened installation.
Additionally, version 0.3.1 of the TrouSerS repository was installed. The entire architecture was booted using TrustedGRUB version 1.1.3. The SE-Linux system was tested using the security policy targeted, since this is the typical policy used by desktop systems.
Appendix C describes the general procedure for turning a regular Gentoo system into a hardened SE-Linux system.

3.2.1.2. The Xen Hypervisor

Xen is free Virtual Machine Monitor software, also known as a hypervisor [3, 12]. In contrast to other VMM's such as VMware, Xen is implemented directly above the hardware layer using a monolithic approach. Xen does not include any hardware device drivers. Instead, it uses the device drivers of a privileged Linux instance, Dom-0, running

22 See http://www.gentoo.org/proj/en/hardened/selinux/index.xml

on top of it. Dom-0 also includes management software (Xen Tools).
In parallel to Dom-0, other virtual machines (i.e. guests) such as Dom-U can be executed in parallel. The Xen-based architecture is shown in figure 3.5.

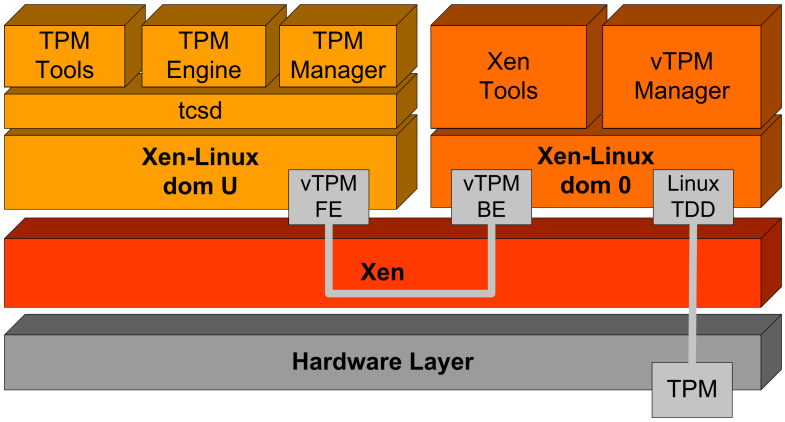


Figure 3.5.: Xen-based architecture.

Since version 3.0, Xen has provided initial steps towards TPM virtualization [4]. Virtual TPM devices are exported to any guest domain using appropriate front ends (i.e. vTPM FE) and a back end (i.e. vTPM BE). The communication between them is based on the Xen Bus. The vTPM front ends are managed through a vTPM Manager implemented by the `vtpm_managerd` daemon. The state of a virtual TPM in a guest domain is bound to the actual hardware TPM, which is accessible through ordinary TPM device drivers inside the Dom-0 Linux. The Dom-U uses its own TrouSerS TSS instance on top of the vTPM driver while the vTPM Manager of Dom-0 directly accesses the TPM through the TPM driver.

3.2.1.3. The Turaya Security Kernel

Turaya is a security kernel providing strong isolation and multilaterally secure policy enforcement of legacy applications [16, 1]. As illustrated in figure 3.6, a Turaya-based software architecture consists of the following three layers: (i) a hardware layer, including conventional components such as memory, CPU, devices and the TPM; (ii) the Turaya security kernel, including a hypervisor layer and a trusted software layer; (iii) applications and legacy operating systems that are executed in parallel and isolated from each other.

Hypervisor layer. The hypervisor layer of the Turaya security kernel acts as a traditional VMM by managing the hardware resources and providing basic virtualization support. Due to the modular concept of the Turaya architecture, different VMM s and microkernels can be used for the hypervisor layer. In the following test scenario, an instance of the Turaya security kernel based on the L4 microkernel (cmp. figure 3.7) is used. A microkernel is a minimal operating system kernel which, in its pure form, provides no operating system services at all; only the mechanisms needed to implement these

services, such as low-level address space management, thread management and Inter-Process Communication (IPC) are provided. On top of the microkernel, fundamental services dedicated to resources are executed. These include services for the management of processes, memory and interrupts as well as those providing device drivers and enforcing system-wide security policies.

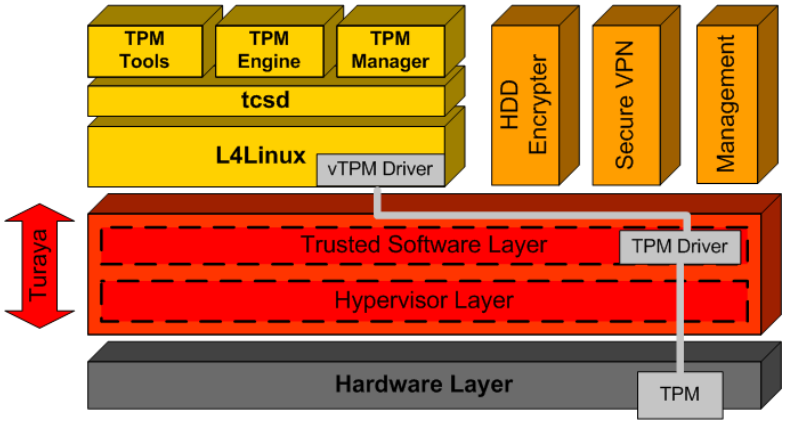


Figure 3.6.: Turaya-based Trusted Computing architecture.

Trusted software layer. Above the hypervisor layer, the trusted software layer provides high-level security services. The secure user interface (SecureGUI), for example, provides a Trusted Path between application and user. The compartment manager manages the creation, update and deletion of compartments. Moreover, it controls which compartments are allowed to be installed and enforces the mandatory security policy. The storage manager provides persistent storage to be used by the other compartments while preserving the integrity, confidentiality, availability and freshness of the stored data. Moreover, the storage manager enforces strong isolation by binding the stored data to the compartment configuration.
The Turaya-based test environment includes a Linux instance running on top of the trusted software layer. This Linux instance uses a TPM driver stub (vTPM Driver) to access the TPM multiplexer (TPM Driver) that is part of the Trusted Software Layer. The Linux vTPM driver and the Turaya TPM driver communicate using the IPC mechanism provided by the underlying microkernel. On top of the Linux vTPM driver, TrouSerS, TPM Tools and the TPM Manager are executed. The entire architecture is booted using TrustedGRUB version 1.1.3.

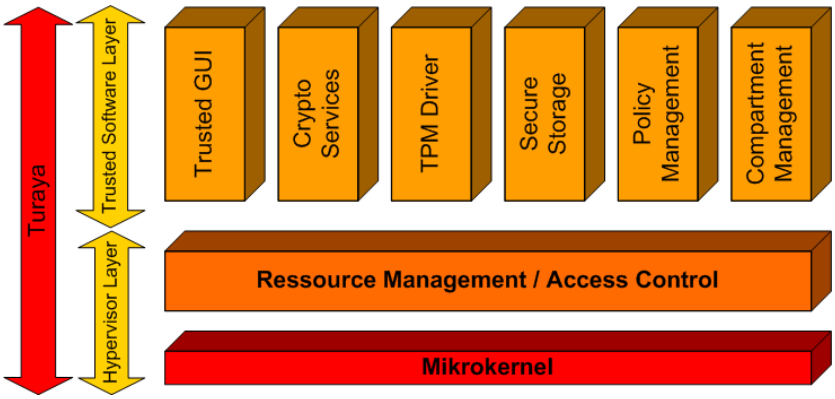


Figure 3.7.: Components of the Turaya security layer.

3.2.2. Test Cases

Based on the software architectures described above, the following functional tests were performed to check the interoperability:

- Tests based on the TPM Manager
  - TC.1 Taking ownership
  - TC.2 Changing owner password
  - TC.3 Deactivating the TPM
  - TC.4 Disabling the TPM
  - TC.5 Enabling the TPM

- Tests based on the TrouSerS TPM Tools
  - TC.6 Sealing a file
  - TC.7 Unsealing a file

- Tests based on the TrouSerS OpenSSL TPM Engine
  - TC.8 Creating a TPM-shielded cryptographic key
  - TC.9 Signing a file using the TPM-shielded signature key

Note. Since TPM Tools include the command `tpm_sealdata`, which is used to seal a file, but does not provide an appropriate tool to unseal it, the command `tpm_unseal` was developed to test whether the sealing functionality worked correctly. The source code of that command is printed in Appendix A.

3.3. Interoperability Tests

The following chapter summarizes results of the interoperability tests performed, including a discussion of issues, pitfalls and appropriate solutions.

3.3.1. SE-Linux-based Architecture

As described in Section 3.2.1.1, an SE-Linux-based computing platform enforces the least privilege principle by granting only those permissions truly needed by a daemon or application.
With the installed Gentoo SE-Linux system in the enforcement mode "targeted" as described in Section 3.2.1.1 and Appendix C, all tests described above succeeded. To perform the tests based on the TPM Manager, however, a graphical user interface (i.e. KDE) and the TSS Core Service daemon had to be started. The graphical system was only able to start after installing the SE-Linux policy `sec-policy/selinux-desktop`. The `tcsd` started successfully without any changes.
When executing SE-Linux in the enforcement mode "strict", it is possible to start the `tcsd` after installing additional policy sources. The necessary rules come with the TrouSerS distribution (written for Fedora Core 5) and are listed in Listing 3.2 and Listing 3.4. The installation routine for Fedora Core is described in Listing 3.3. Within a Gentoo Linux, the policy routines have changed since 2006, so the rules are not set via Makefiles anymore, but will be compiled as loadable modules. Therefore, new utilites exist under Gentoo called `checkmodule`, `semodule`, `semodule_package` and `audit2allow`. One easy (but not very secure) way would be to let the necessary rules been generated automatically by `audit2allow`. In order to do so, reboot the SE-Linux and try to start TrouSerS. The startup will fail, but it is possible to audit the required permissions and to create a new ruleset for the `tcsd` by:

```
// Creating ruleset:
# audit2allow -d -M tcsd
// Installing new ruleset:
# semodule -i tcsd.pp
```

Afterwards, TrouSerS `tcsd` and the TPM-Tools work without problems. Please keep in mind that this is a very generic way to create valid rulesets for Gentoo. `audit2allow` will only check which permissions were denied and will grant the required permissions automatically.
But these permissions might be too loose, so it would be more appropriate to adapt the more specific Fedora Makefile-ruleset to the Gentoo modular one.
Although it is possible to load a TPM device driver, start TrouSerS and use the TPM-Tools within the strict-mode of SE-Linux, it is not possible to use the TPM-Manager, since it is not possible to login to the XServer. This is due to different security contexts of the XServer and the user, which results in a security violation while accessing the shadow file to verify the password of the user[23].

Note: Since Gentoo uses `udev` for the management of the device-nodes in `/dev/`, SE-Linux doesn't have the correct permissions on startup in strict-mode. This is due to the fact

23 It is questionable whether it even makes sense to allow the running of an XServer inside a strict SE-Linux-protected environment.

that at the point where SE-Linux starts, the device-nodes are still static and have not been generated by udev. Therefore, SE-Linux will prevent the user to login, since the access to the console and getty files are not granted. This issue can be solved by switching Gentoo to use static device nodes only. In order to do so, `set RC_DEVICES=static` in `/etc/conf.d/rc`. Afterwards, reboot your system and relabel the files again with

```
rlpkg -a -r.
```

Since udev is not running anymore, you have to create the TPM-device node manually via

```
mknod /dev/tpm0 c 10 224
```

and set the permissions via

```
chown tss:tss /dev/tpm0.
```

Additionally, the global boolean `global_ssp` has to be true, but is false on default within Gentoo. Enable it via

```
setsebool -P global_ssp 1.
```

Listing 3.2: trousers.fc security policy for Fedora

```
/usr/sbin/tcsd system_u:object_r:tcsd_exec_t
/etc/tcsd.conf system_u:object_r:tcsd_config_t
/var/lib/tpm(/.*)? system_u:object_r:tcsd_readwrite_t
/dev/tpm(.*) system_u:object_r:tcsd_device_t
```

After successfully loading the security policies, the result should look like this:

```
# ls -Z /dev/tpm*
crw-rw---- root root system_u:object_r:tcsd_device_t  /dev/tpm0
# ps -Zef | grep tcsd
root:system_r:tcsd_t root 16362 1 0 15:10 ?00:00:00 /usr/sbin/tcsd
```

Listing 3.3: Installing TrouSerS SE-Linux security policies within Fedora

```
# cp ./dist/fedora/trousers.te \
  /etc/selinux/targeted/src/policy/domains/program
# cp ./dist/fedora/trousers.fc \
  /etc/selinux/targeted/src/policy/file_contexts/program
# cd / etc / selinux / targeted / src / policy
# make clean
# make reload
# make install
# make relabel
```

Listing 3.4: trousers.te security policy for Fedora

```
type tcsd_device_t , device_type , dev_fs ;
type tcsd_readwrite_t , file_type ;
type tcsd_config_t , file_type , sysadmfile ;
daemon_domain ( tcsd , ‘ )
general_domain_access ( tcsd_t )
allow unconfined_t tcsd_t : process transition ;
type_transition unconfined_t tcsd_exec_t : process tcsd_t ;
allow tcsd_t tcsd_exec_t : dir r_dir_perms ;
allow tcsd_t etc_t : file { read getattr lock ioctl } ;
allow tcsd_t etc_t : lnk_file { read getattr } ;
allow tcsd_t devtty_t : chr_file { ioctl read getattr lock write append } ;
allow tcsd_t devpts_t : chr_file { ioctl read getattr lock write append } ;
can_network ( tcsd_t )
read_sysctl ( tcsd_t , full )
r_dir_file ( tcsd_t , usr_t )
r_dir_file ( tcsd_t , tcsd_config_t )
rw_dir_file ( tcsd_t , tcsd_readwrite_t )
allow tcsd_t tcsd_readwrite_t : file { setattr } ;
allow tcsd_t tcsd_readwrite_t : dir { setattr } ;
allow tcsd_t tcsd_device_t : chr_file { ioctl read getattr lock write append } ;
allow tcsd_t { random_device_t } : chr_file { read getattr } ;
allow tcsd_t lib_t : dir r_dir_perms ;
allow tcsd_t lib_t : file { rx_file_perms execmod } ;
allow tcsd_t lib_t : lnk_file r_file_perms ;
allow tcsd_t lib_t : file { rx_file_perms execmod } ;
allow tcsd_t lib_t : lnk_file r_file_perms ;
allow tcsd_t lib_t : file { rx_file_perms execmod } ;
allow tcsd_t lib_t : lnk_file r_file_perms ;
allow tcsd_t var_lib_t : dir r_dir_perms ;
allow tcsd_t var_lib_t : file { rx_file_perms execmod } ;
allow tcsd_t var_lib_t : lnk_file r_file_perms ;
allow tcsd_t port_type : tcp_socket { send_msg recv_msg name_bind } ;
allow tcsd_t self : capability { chown net_bind_service dac_override fowner
fsetid
```

3.3.2. Xen-based Architecture

The current implementation of the vTPM-Manager implements the interface and commands of TPM specification 1.1b. Due to some deleted commands in the v1.2 TPM specification, it is currently not possible to use it with version 1.2 TPMs. Therefore, it is also not possible to virtualize the TPM using a version 1.2 TPM/TSS, which lack the fundamental requirements to execute the interoperability tests listed in Section 3.2.2.
Therefore, the interoperability tests were executed on an IBM Thinkpad T40 including an Atmel TPM implementing TPM Specification v1.1b. On top of this platform, Xen v3.1.0 was installed. Since version 1.2 TPMs are not currently supported by the vTPM Manager, TrouSerS v0.2.9 was installed in dom-U. The entire architecture was booted using TrustedGRUB version 1.1.3.

The following summarizes the results of the interoperability tests performed. Included is a discussion of issues, pitfalls and appropriate solutions.

- **vTPM and vTPM Manager** *(Dom-0)*:
  The vTPM Manager inside Xen was started successfully. Whenever a guest domain is booted within Xen, a new vTPM is created and connected to the guest OS. Since this vTPM is newly created upon every start of the guest domain, one has to exe cute `tpm_takeownership` every time the guest machine is booted in order to bring the vTPM into an owned state. Once the vTPM Manager is started correctly, and the corresponding Linux vTPM device driver is loaded, the vTPM behaves like a normal v1.1b TPM.
- **TrouSerS** *(Dom-U)*:
  The tcsd of TrouSerS inside the guest domain started as expected.
- **TPM Tools** *(Dom-U)*:
  Due to some bugs in earlier versions of TrouSerS, above all in version 0.2.9 which was used for the tests under Xen, one is not able to execute `tpm_sealdata` correctly since the authentication data inside tpm_sealdata is not handled correctly. In addition, resources (e.g. sessions) opened by the vTPM are not closed correctly within the TPM hardware. This leads to insufficient resources inside the TPM after just a few commands. The resources are not freed until the machine is rebooted.
- **TPM Manager** *(Dom-U)*:
  The TPM Manager running within a Xen-virtualized guest domain was not able to find out the true state of the vTPM and therefore assumed that the vTPM is disabled.
- **OpenSSL TPM Engine** (*Dom-U*):
  Testing produced successful results.

3.3.3. Turaya-based Architecture

In contrast to the vTPM available under Xen, the TPM framework of the Turaya security kernel does not provide a virtual TPM that can be used by different legacy OS instances in parallel. Instead, Turaya maps the functionality offered by the TPM to different security services providing a higher-level abstraction of the provided functionality. For example, the monotonic counters provided by a TPM since version 1.2 are mapped to the Secure Storage service providing fresh persistent storage. Although this design allows the realization of security-critical applications very efficiently, it currently does not allow use of a virtual TPM from different legacy operating systems. However, according to the Turaya roadmap, a virtual TPM implementation based on the provided security services is currently under development.

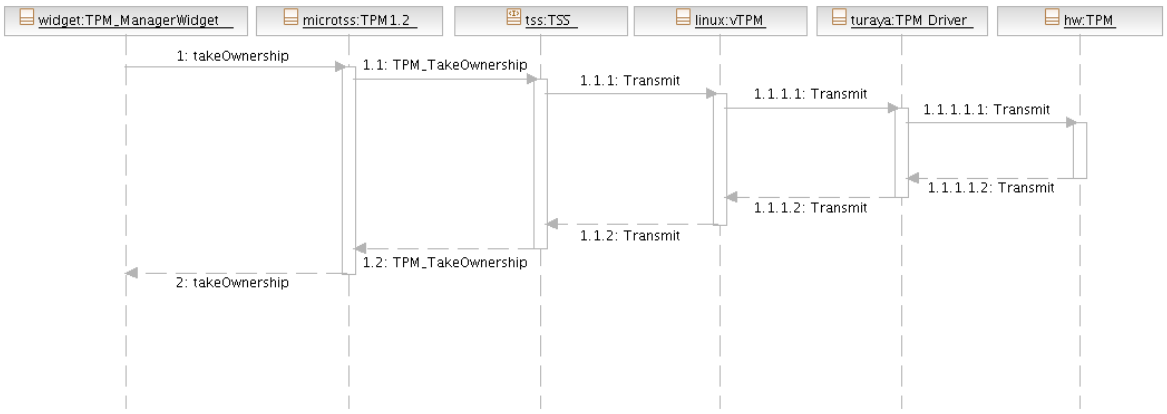


Figure 3.8.: Message flow after invocation of "Take Ownership" from a Turaya-based system.

Figure 3.8 is a somewhat simplified illustration of the flow of messages after a “Take Ownership” operation is invoked by the TPM Manager on top of the Turaya security kernel.
The Linux TPM driver simply forwards the TPM command to the Turaya TPM driver, which itself forwards the command on to the physical TPM.
Since the TPM driver of Turaya - here implemented as an isolated service within the Turaya security layer (cmp. figure 3.7) - directly forwards TPM commands to the physical TPM, no interoperability problems were expected. In fact, the interoperability tests based on TPM Tools, the TPM Manager, and the OpenSSL TPM engine all ran successfully.

4. Conclusion

This study provides an overview of existing open source software supporting Trusted Computing technology as specified by the Trusted Computing Group, evaluates their compliance and identifies shortcomings and missing components required to build a Trusted Computing platform.
The software components that have been analyzed and tested are various TCG-enabled boot loaders, the Linux TPM Device Driver, the TrouSerS TCG Software Stack, the TrouSerS TPM Tools, the OpenSSL TPM engine and the TPM Manager.
Lastly, the interoperability and functionality of the here introduced software running on the following security architectures have been evaluated: Security-Enhanced Linux, the Xen hypervisor and the Turaya security kernel.

4.1. Trusted Computing Building Blocks

Some functionality in the evaluated open source solutions are either not available or not working properly. However, this study has shown that the most important building blocks, namely a TCG-compliant boot loader, a TPM driver framework and an implementation of the TCG Software Stack, are available and robust enough to be used in a wide variety of security-critical services and applications.
However, it is apparent that, until now, no application exists that makes use of this technology. Even the simplest applications, e.g., the use of the TPM to create random numbers within Linux, have not been applied yet. In theory, the OpenSSL engine concept, together with the OpenSSL TPM engine, should provide an easy way of migrating existing applications to use the additional features offered by a TPM. This is because the OpenSSL engine concept allows extension of the OpenSSL framework with additional cryptographic modules. One result of our experiences is that, in practice, the engine concept of OpenSSL does not offer such an easy migration. Instead, use of the OpenSSL TPM engine concepts requires too many modifications of application code.

4.2. Trusted Computing Architectures

Analysis of Trusted Computing architectures based on SE-Linux, Xen and Turaya has shown that the interoperability of existing Trusted Computing solutions is high enough to realize TC-enabled applications on top of them. In the following, we discuss the open issues identified during this analysis:

- Xen
  Using the development branch of the TrouSerS TSS stack, it was possible to realize all selected use cases using a version 1.2 TPM. The only exception at the time of writing is the vTPM implementation of the Xen hypervisor, which is not stable yet. During our tests, the vTPM crashed very often and losts its internal state whenever the guest virtual machine was rebooted.
- Turaya
  Based on the Turaya security kernel, it was also possible to realize all use cases. However, our analysis has shown that the Turaya security kernel currently lacks sup port of the vTPM implementation. Although a TPM driver exists to connect one virtual machine with a TPM, it is currently not possible to use a TPM within several virtual machines in parallel. According to the Turaya development team, a virtual TPM implementation is currently under development.
- SE-Linux
  The interoperability tests based on SE-Linux showed on the one hand that the SE-Linux support of common Linux distributions is good enough to perform the selected test cases. In contrast to the 'strict' enforcement mode that protects all system services, the mode 'targeted' only protects a limited number of system services such as the HTTP daemon.

Although all three Trusted Computing architectures can be used to realize applications with Trusted Computing support, a sound and stable TC architecture supporting TC technology up to the application layer is currently not available yet. The EMSCB project[24] and the OpenTC project[25] aim to develop such an architecture based on the Xen hypervisor and the Turaya security kernel. Today, these architectures can already be used for specific applications and scenarios, but they are still not ready for general-purpose use in the mass market.
Based on Linux, only the Integrity Measurement Architecture [17] developed by IBM exists. It allows a remote party to attest the system configuration, including all loaded configuration files and shared libraries, but it also has some important drawbacks. IMA is an extension to Linux that inserts measurement hooks in functions relevant for loading code. Whenever a kernel module, application, or configuration file is loaded, the file is measured and the resulting hash value used to extend a PCR. In this way, IMA extends the measurement chain from the BIOS and boot loader up to the application level. The main purpose of IMA is to analyze the trustworthiness of loaded content according to the measurement log provided during the attestation protocol. However, IMA does not allow the attestation of single applications as opposed to the entire computing platform, including the operating system and all loaded applications. Moreover, it does not allow the binding or sealing of data in certain applications and configurations since the PCR values change whenever a new file is loaded. This causes sealed data to be unavailable. The status of the PCR further depends on the sequence of files loaded. If the sequence changes, the PCR

24 http://www.emscb.org
25 http://www.opentc.net

values are affected.

4.3. Open Issues

Based on the evaluation result of this study, the following improvements are required to initiate a wider use of Trusted Computing technology.

4.3.1. Cryptographic API with TC support

In the short term, existing security-critical applications, such as VPN modules or hard disk encrypters, should be extended by TPM support to benefit from the extended features of Trusted Computing technology. However, extending every existing security-relevant application by Trusted Computing and TSS support does not appear to be realistic, as it would require very significant resources. This is all the more the case because the TSS interface and according TrouSerS implementation themselves are under heavy development.
Under Microsoft Windows, applications can use a cryptographic interface (MS-CAPI) permitting the use of TPM functions. However, such a unique cryptographic interface does currently not exist under Linux. At the moment, the only alternative is the OpenSSL engine interface, since OpenSSL is a quasi-standard used by many security-relevant applications. Figure 4.1 shows how a cryptographic API can be used to extend conventional applications to use the TSS.

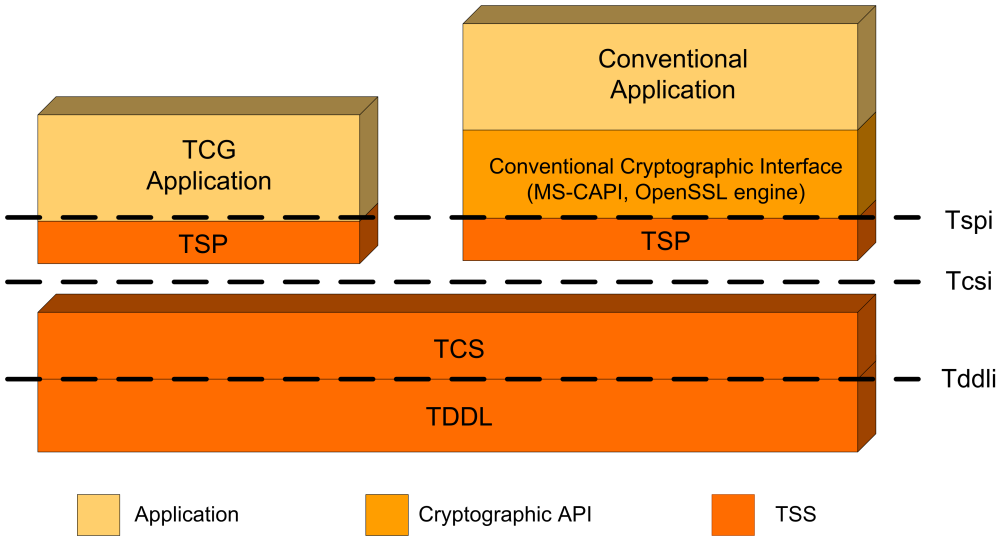


Figure 4.1.: Usage of TSS services based on a conventional cryptographic interface compared to native TCG-enabled applications making direct use of the TSP.

As discussed in Section 2.6, making use of the OpenSSL engine is currently not as easy as it should be, as every application has to be extended by OpenSSL engine support separately.
According to our analysis based on various OpenSSL-supporting applications (e.g. racoon[26] and XSupplicant[27]), such an extension would require much effort and collaboration with the developers of the application.
In the long term, however, it would be more efficient to improve the OpenSSL engine concept such that it works transparently without the need to modify the applications themselves. This would allow the use of a TPM by all applications supporting OpenSSL.
Without the possibility of using TPM functions based on a common cryptographic interface, TPM support of Linux (and alternative open source solutions in general) will not progress. As a result, open source solutions will continue to fall behind proprietary solutions.

4.3.2. Management Solutions

Another important aspect of TC-based products is the development of management concepts and tools to bypass typical shortcomings of TC technology.
Modification of a measurements binary, for example, changes the PCR values such that sealed data cannot be accessed anymore. Whereas this is beneficial in the case of malicious modification, system updates and security patches (of either software or firmware) necessary to keep an IT system up-to-date and secure, may be hindered. As long as TC technology identifies binaries by their hash values, reliable management tools are required to help administrators update sealed data in the case of system modifications. Moreover, these tools must be integrated into the software management software of operating system distributions to enable the detection of modifications to critical system components.
Another example of required concepts and tools involves the development of appropriate solutions to manage and backup keys shielded by the TPM. In the case of a hardware failure (e.g. of the motherboard or the TPM itself), users need a reliable mechanism to restore their data. Since TPM-shielded keys never leave the TPM as plain text, these backups would need to be created and managed by application-level software.
A further important advancement would be the realization of platform migration strategies [10]. If users want to move their data to a newer platform including another TPM (which may be from another vendor), their TPM-protected data has to be migrated as well. The migration concept specified by the TCG is only optional for vendors and, to our knowledge, yet to be implemented by any of the TPM vendors. Moreover, it only allows migration to a system secured by a TPM from the same manufacturer. To bypass these restrictions, appropriate tools are required to realize data migration on the software level.
To our knowledge, none of these tools, which are essential to provide a reliable and secure platform, currently exist.

26 http://ipsec-tools.sourceforge.net/
27 http://open1x.sourceforge.net/

4.3.3. Infrastructural Solutions

For the long term, Trusted Computing concepts need to be realized that improve the security, reliability, and trustworthiness of existing IT infrastructures while providing interoperability with existing solutions. Possible approaches are the combination of virtualization technology, Trusted Computing and security kernels, as offered by Turaya and OpenTC (see Section 3.2.1). Alternatively, existing IT infrastructures such as Linux can be extended by Trusted Computing (TC) technology, as demonstrated by the Integrity Measurement Architecture discussed in Section 4.2.

However, these operating system architectures currently do not (and, in fact, are not able to) solve all problems. Several open issues exist with regard to infrastructure and management, in particular:

The TCG specification specifies, for example, that so-called “platform certificates” issued by the platform integrator to state that the TPM-enabled platform has been manufactured according to the TPM specifications. These certificates have to be evaluated during an attestation or Attestation Identity Key (AIK) certification process to determine the trustworthiness of that platform. By now, platform vendors do not provide platform certificates.

Another open issue that has only been considered in part by the Trusted Network Connect (TNC) specification [25] is the question of how the extensive set of possible binary configurations should be managed. Considering the huge number of potential system configurations, software updates, patches and compiler versions, the number of configurations is huge already and would grow exponentially. Moreover, many applications and operating systems such as Linux are compiled locally, which further increases the number of configurations since many compilers insert the compilation time and similar dynamic information into the binary.

As far as attestation, binding and sealing are concerned, it is often criticized that the TCG approach leaks more information about a platform than necessary. An adversary can, on the one hand, observe binary measurements to identify an operating system or application so as to optimize attacks (keyword: fingerprinting). On the other hand, these binary values also allow vendors to discriminate against certain binary configurations (e.g. refusing to support specific operating systems or applications). Current research on *property-based* and *semantic attestation* [15, 8] seeks solutions that only attest certain aspects of a platform. Proposed solutions are not yet ready to be used for the mass market.

Open issues in the field of Trusted Computing are currently being addressed in various stages ranging from conception to solution architecture and design on towards actual implementation. Despite the shown difficulties, the introduced specifications of the TCG for the TPM incl. the TSS, in context with the described security architectures, offer good potential for the realization of trustworthy computing platforms. As it was shown, first existing software solutions exist that support this technology. Although those are not yet mature enough, they are tending towards the right direction.

Appendices

Appendix A. TPM_Unseal

The following source code defines the `tpm_unseal` used to unseal a file using a TPM. The required Unseal-function itself is delivered in a library of TrouSerS and can be used by including the header file `tpm_unseal.h`.

```
/*
 * tpm_unseal - A small tool to unseal files using a TPM.
 *
 * Based on:
 * (c) 2007 Gianluca Ramunno (TORSEC group - Politecnico di
 * Torino)
 * This software is released under GPL licence v2.
 */
#include <stdio.h>
#include <tpm_unseal.h>
void usage () {
  printf ("tpm_unseal <sealed_file_name>\n");
  printf (" Upon successful unsealing, the file will be sent to
the standard output\n");
}
int main (int argc, char **argv)
{
  int res, retcode=0, size;
  unsigned char *data;
  if (argc !=2) {
    usage();
    return(2);
  }
  if ((res = tpmUnsealfile( argv[1], &data, &size )) == 0)
    write (1, data, (size_t)size);
  else
    retcode = 1;
  fprintf(stderr, "%s\n", tpmUnsealStrerror(res));
  tpmUnsealShred(data, size);
  return (retcode);
}
```

Compile by entering:

```
gcc -I/usr/include/tpm_tools/ tpm_unseal.c -ltpm_unseal \
-o tpm_unseal.
```

Appendix B. Patches for TPM Tools

The following patches set the `TSS_WELL_KNOWN_SECRET` within the `tpm_sealdata`-function and the `tpm_unsealfile`-function.

Patch for `tpm_sealdata.c`

```
~ $ cat tpm_sealdata.c.diff
--- tpm-tools-1.3.0/src/cmds/tpm_sealdata.c
+++ tpm-tools-1.3.0-patched/src/cmds/tpm_sealdata.c

@@ -118,6 +118,7 @@ int main(int argc, char **argv)
            TSS_KEY_VOLATILE | TSS_KEY_AUTHORIZATION |
            TSS_KEY_NOT_MIGRATABLE;
        TSS_HPOLICY hSrkPolicy;
+  BYTE well_known_secret[TPM_SHA1_160_HASH_LEN] = TSS_WELL_KNOWN_SECRET;

   BIO *bin = NULL, *bdata=NULL, *b64=NULL;

@@ -169,7 +170,8 @@ int main(int argc, char **argv)
   if (policyGet(hSrk, &hSrkPolicy) != TSS_SUCCESS)
               goto out_close;
-  if (policySetSecret(hSrkPolicy, 0, NULL) != TSS_SUCCESS)
+  if (policySetSecret(hSrkPolicy,TPM_SHA1_160_HASH_LEN,
+       well_known_secret)!=TSS_SUCCESS)
+//if (policySetSecret(hSrkPolicy, 0, NULL) != TSS_SUCCESS)
               goto out_close;
        /* Build an RSA key object that will be created by the TPM
```

Patch for `tpm_unseal.c`

```
~ $ cat tpm_unseal.c.diff
--- tpm-tools-1.3.0/lib/tpm_unseal.c
+++ tpm-tools-1.3.0-patched/lib/tpm_unseal.c

@@ -81,6 +81,7 @@ int tpmUnsealfile( char* fname, unsigned
        TSS_HPOLICY hPolicy;
        UINT32 symKeyLen;
        BYTE *symKey;
+       BYTE well_known_secret[TPM_SHA1_160_HASH_LEN] =
            TSS_WELL_KNOWN_SECRET;
        unsigned char* res_data = NULL;
        int res_size = 0;

@@ -332,7 +333,8 @@ int tpmUnsealfile( char* fname, unsigned
               goto tss_out;
        }
- if ((rc=Tspi_Policy_SetSecret(hPolicy, TSS_SECRET_MODE_PLAIN, 0, NULL))
+//if ((rc=Tspi_Policy_SetSecret(hPolicy, TSS_SECRET_MODE_PLAIN, 0,
      NULL))
```

```
+ if ((rc=Tspi_Policy_SetSecret(hPolicy, TSS_SECRET_MODE_SHA1,
+ TPM_SHA1_160_HASH_LEN, well_known_secret))
                                          != TSS_SUCCESS) {
                tpm_errno = ETSPIPOLSS;
                goto tss_out;
```

Appendix C. SE-Linux Installation

The following steps are required to turn a regular Gentoo system into a hardened SE-Linux:

1. Compile the SE-Linux enhanced kernel
2. Switch the system profile to enable hardened support
3. Install required userland utilities
4. Configure the system environment to use SE-Linux

Compiling the SE-Linux enhanced kernel. The following kernel parameters need to be configured inside Linux in order to enable NSA SE-Linux support.
Note that only four file systems: `ext2, ext3, jfs` and `xfs` are currently supported. In addition, make sure that the Linux version of Adobe Acrobat acroread is NOT installed, since this application has led to a text relocation problem when re-labeling the file system.

Switching the system profile. One has to switch to a hardened Gentoo profile to enable certain security use flags (`e.g. USE = selinux`):

Installing necessary userland utilities. In order to use SE-Linux system-wide, one has to install additional libraries, policies and utilities by entering:

```
# emerge sysvinit checkpolicy policycoreutils selinux-base-policy.
```

Afterwards, it is necessary to re-compile the installed applications to use their own SE-Linux policies by invoking:

```
# emerge --newuse world
```

Configuring the system environment to use SE-Linux. The configuration of SE-Linux is carried out in several steps:

1. Choose the Policy Type in `/etc/selinux/config`

```
# This file controls the state of SELinux on the system on
# boot. SELINUX can take one of these three values:
#  enforcing - SELinux security policy is enforced.
#  permissive - SELinux prints warnings instead of enforcing.
#  disabled - No SELinux policy is loaded.
SELINUX=enforcing

# SELINUXTYPE can take one of these two values:
#   targeted - Only targeted network daemons are protected.
```

```
#   strict - Full SELinux protection.
SELINUXTYPE=targeted
```

2. Reboot into SE-Linux

3. Modify `/etc/fstab`

```
none /selinux selinuxfs defaults 0 0
```

4. Label file systems with

```
# rlpkg -a -r
```

Listing C.2: Switching to a hardened Gentoo profile

```
# rm -f /etc/make.profile

x86 ( hardened ) :
# ln -sf /usr/portage/profiles/selinux/2007.0/x86/hardened/ \
  etc/make.profile

AMD64 ( hardened ) :
# ln -sf /usr/portage/profiles/selinux/2007.0/amd64/hardened/ \
  etc/make.profile
```

Listing C.1: SE-Linux kernel configuration

```
cd /usr/src/linux-2.6.22-hardened-r7
make menuconfig

Under Code maturity level options
[*] Prompt for development and / or incomplete code / drivers

Under General setup
[*] Auditing support
[*]     Enable system-call auditing support

Under file systems
<*> Second extended fs support ( If using ext2 )
[*]     Ext2 extended attributes
[ ]        Ext2 POSIX Access Control Lists
[*]        Ext2 Security Labels
<*> Ext3 journalling file system support ( If using ext3 )
[*]     Ext3 extended attributes
[ ]        Ext3 POSIX Access Control Lists
[*]        Ext3 Security labels
<*> JFS filesystem support ( If using JFS )
[ ]     JFS POSIX Access Control Lists
[*]     JFS Security Labels
<*> XFS filesystem support ( If using XFS )
[ ]     Realtime support ( EXPERIMENTAL )
[ ]     Quota support
[ ]     ACL support
[*]     Security Labels

Under file systems -> Pseudo file systems
[ ] /dev file system support ( EXPERIMENTAL )
[*]     /dev/pts Extended Attributes
[*]        /dev/pts Security Labels
[*] Virtual memory file system support ( former shm fs )
[*]     tmpfs Extended Attributes
[*]        tmpfs Security Labels

Under Security options
[*] Enable different security models
[*]     Socket and Networking Security Hooks
<*>     Default Linux Capabilities
[*] NSA SELinux Support
[ ]     NSA SELinux boot parameter
[ ]     NSA SELinux runtime disable
[*]     NSA SELinux Development Support
[ ]     NSA SELinux AVC Statistics
(1)     NSA SELinux checkreqprot default value
[ ]     NSA SELinux enable new secmark network controls by default
[ ]     NSA SELinux maximum supported policy format version
```

Appendix D. Creative Commons License by ND

CREATIVE COMMONS CORPORATION IS NOT A LAW fiRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License: Attribution-NoDerivs 3.0 Unported

THE WORK (AS DEfiNED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

a. **"Adaptation"** means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

b. **"Collection"** means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

c. **"Distribute"** means to make available to the public the original and copies of the Work through sale or other transfer of ownership.

d. **"Licensor"** means the individual, individuals, entity or entities that offer(s) the Work under

the terms of this License.

e. **"Original Author"** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

f. **"Work"** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

g. **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

h. **"Publicly Perform"** means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

i. **"Reproduce"** means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to

Reproduce the Work as incorporated in the Collections; and,

b. to Distribute and Publicly Perform the Work including as incorporated in Collections.
c. For the avoidance of doubt:

    i. **Non-waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
    ii. **Waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,
    iii. **Voluntary License Schemes**. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Adaptations. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(b), as requested.
b. If You Distribute, or Publicly Perform the Work or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright

notice or licensing information for the Work. The credit required by this Section 4(b) may be implemented in any reasonable manner; provided, however, that in the case of a Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

c. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR offers THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, fiTNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effectunless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
e. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effectin the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at http://creativecommons.org/.

Bibliography

[1] Ammar Alkassar, Michael Scheibel, Ahmad-Reza Sadeghi, Christian Stüble, and Marcel Winandy. Security architecture for device encryption and VPN. In Sachar Paulus, Norbert Pohlmann, and Helmut Reimer, editors, Information Security Solutions Europe (ISSE 2006), pages 54–63. Vieweg Verlag, 2006.

[2] AMD. AMD64 virtualization codenamed "Pacifica" technology - secure virtual machine architecture reference manual. Technical Report Publication Number 33047, Revision 3.01, AMD, May 2005.

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Adrew Warfield. Xen and the art of virtualization. In Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03), Bolton Landing, NY, USA, October 2003. ACM.

[4] Stefan Berger, Ramon Caceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vTPM: Virtualizing the Trusted Platform Module. In Proceedings of the 15th USENIX Security Symposium, pages 305–320. USENIX, August 2006.

[5] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. Rz 3540, IBM Research, March 2004.

[6] Trusted Computing Group. TCG software stack specification http://trustedcomputinggroup.org, August 2003. Version 1.1.

[7] Trusted Computing Group. TCG software stack specification. http://trustedcomputinggroup.org, January 2006. Version 1.2.

[8] Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation: A virtual machine directed approach to trusted computing. In USENIX Virtual Machine Research and Technology Symposium, May 2004. also Technical Report No. 03-20, School of Information and Computer Science, University of California, Irvine; October 2003.

[9] IBM Inc. The TrouSerS webpage. http://trousers.sourceforge.net/, July 2007.

[10] Ulrich Kühn, Klaus Kursawe, Stefan Lucks, Ahmad-Reza Sadeghi, and Christian Stüble. Secure data management in trusted computing. In Josyula R. Rao and Berk Sunar, editors, Cryptographic Hardware and Embedded Systems - CHES 2005,

volume 3659 of Lecture Notes in Computer Science, pages 324–338. Springer-Verlag, Berlin Germany, 2005.

[11] Ulrich Kühn, Marcel Selhorst, and Christian Stüble. Realizing property-based attestation and sealing with commonly available hard- and software. In Proceedings of the 2nd ACM Workshop on Scalable Trusted Computing (STC'07). ACM Press, 2007.

[12] Aravind Menon, Alan L. Cox, and Willy Zwaenepoel. Optimizing network virtualization in Xen. In USENIX Annual Technical Conference, Boston, MA, 2006.

[13] The OpenSSL Project. The OpenSSL webpage. http://www.openssl.org/, July 2007.

[14] The TPM Manager project. The TPM Manager website. http://www.sourceforge.net/projects/tpmmanager/, July 2007.

[15] Ahmad-Reza Sadeghi and Christian St¨ble. Property-based attestation for computing platforms: Caring about properties, not mechanisms. In The 2004 New Security Paradigms Workshop, Virginia Beach, VA, USA, September 2004. ACM SIGSAC, ACM Press.

[16] Ahmad-Reza Sadeghi, Christian Stüble, and Norbert Pohlmann. European multilateral secure computing base - open trusted computing for you and me. Datenschutz und Datensicherheit DuD, Verlag Friedrich Vieweg & Sohn, Wiesbaden, 28(9):548–554, 2004.

[17] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. 13th Usenix Security Symposium, San Diego, California, pages 223–238, August 2004.

[18] Stephen Smalley. Configuring the SELinux policy. Report #02-007, NAI Labs, February 2002. Revised April 2002.

[19] Stephen Smalley, Chris Vance, and Wayne Salamon. Implementing SELinux as a Linux security module. Report #01-043, NAI Labs, December 2001. Revised May 2002.

[20] Ray Spencer, Stephen Smalley, Peter Loscocco, Mike Hibler, David Andersen, and Jay Lepreau. The flask security architecture: System support for diverse security policies. In Proceedings of the 8th USENIX Security Symposium, pages 123–139, Washington, D.C., USA, August 1999. USENIX.

[21] Christian Stüble and Anoosheh Zaerin. TPM Manager – Requirements, Design, and Implementation. Technical report, Sirrix AG security technologies and Ruhr-

University Bochum, 2006.

[22] Trusted Computing Group. TCG PC-client specific implementation for conventional BIOS version 1.20 fiNAL. Technical report, Trusted Computing Group, Incorporated, July 2005.

[23] Trusted Computing Group. TPM interface specification. Specification Version 1.0, Trusted Computing Group, 2005.

[24] Trusted Computing Group. TPM main specification. Main Specification Version 1.2 rev. 85, Trusted Computing Group, February 2005.

[25] Trusted Computing Group. Trusted network connect. Specification Version 1.2,2007.

[26] Trusted Computing Platform Alliance (TCPA). TCPA PC-specific implementation specification, September 2001. Version 1.00.

[27] Trusted Computing Platform Alliance (TCPA). Main specification, February 2002. Version 1.1b.

[28] Eric A. Young and Tim J. Hudson. The SSLeay project. http://www.columbia.edu/~ariel/ssleay/, July 2007.

List of Acronyms

AIK	Attestation Identity Key
API	Application Programming Interface
BIOS	Basic Input Output System
CHS	Cylinder Head Sector
CRTM	Core Root of Trust for Measurement
DAA	Direct Anonymous Attestation
D-RTM	Dynamic Root of Trust for Measurement
EK	Endorsement Key
GNU	GNU's Not Unix
GPL	GNU General Public License
GRUB	GRand Unified Bootloader
GUI	Graphical User Interface
IMA	Integrity Measurement Architecture
IPC	Inter-Process Communication
LBA	Logical Block Addressing
MAC	Mandatory Access Control
MBR	Master Boot Record
OSLO	Open Secure LOader
PCR	Platform Configuration Register
PrivacyCA	Privacy Certification Authority
RTM	Root of Trust for Measurement
SLOC	Source Lines Of Code
SRK	Storage Root Key
S-RTM	Static Root of Trust for Measurement
TC	Trusted Computing
TCG	Trusted Computing Group
TCS	TSS Core Service
TDD	TPM Device Driver
TDDL	TCG Device Driver Library
TDDLI	TCG Device Driver Library Interface
TIS	TPM Interface Specification

TNC	Trusted Network Connect
TPM	Trusted Platform Module
TSP	TSS Service Provider
TSPI	TSS Service Provider Interface
TSS	TCG Software Stack
UML	Unified Modeling Language
VM	Virtual Machine
VMM	Virtual Machine Monitor

Glossary

Attestation Identity Key

A non-migratable key (Non-Migratable Key) that is created locally by a TPM and provides pseudonymity or anonymity of TPM-secured platforms. The public portion of an AIK is certified by a Privacy Certification Authority (PrivacyCA) stating that this signature key is truly under the control of a secure TPM. In order to negotiate the problem of linked transactions to a certain platform, version 1.2 of the TCG specification defines a cryptographic protocol called Direct Anonymous Attestation (DAA) [5] that eliminates the need for a PrivacyCA.

Basic Input Output System

The code on a PC platform that initializes memory and hardware devices.

Core Root of Trust for Measurement

A PC component specified by the TCG that measures the BIOS before executing it.

Cylinder Head Sector

An early means of giving addresses to each physical block of data on a hard disk drive.

Direct Anonymous Attestation

A cryptographic protocol developed in the context of the TCG specification [5] to avoid third parties link transactions to a certain platform; eliminates the need for a PrivacyCA by using a zero-knowledge protocol.

Dynamic Root of Trust for Measurement

The Dynamic Root of Trust for Measurement (D-RTM) is an RTM, supported by Intel's TXT or AMD's Presidio hardware extension. Dynamic loadable code, running in isolated environments of the CPU, is used as root for a chain of trust. Thus, virtual machines get their own RTM. In case of D-RTM, the PCRs 17 - 22 are used.

Endorsement Key

An asymmetric 2048-Bit RSA-Encryption key, which is unique for every TPM. The EK resides inside the TPM permanently and can be used to authenticate a TPM and its platform.

GNU General Public License

The most widely used license for Free Software.

GNU's Not Unix

The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system. Variants of the GNU operating system, which use the kernel called Linux, are now widely used; though these systems are often referred to as "Linux", they are more accurately called GNU/Linux systems. GNU is a recursive acronym for "GNU's Not Unix"; it is pronounced guh-noo, approximately like canoe. For more information, see http://www.gnu.org/.

GRand Unified Bootloader

A boot loader package from the GNU's Not Unix (GNU) Project implementing the Multiboot Specification, which allows users to have several different operating systems on their computer at once.

Graphical User Interface

A type of user interface allowing people to interact with a computer or computer-controlled device employing graphical icons, visual indicators or special graphical elements called widgets as well as text, labels or text navigation to represent the information and actions available to a user.

Hypervisor

See Virtual Machine Monitor.

Integrity Measurement Architecture

An architecture from IBM that generates verifiable representative information about the software stack running on a GNU Linux operating system, which can be used by remote parties to determine the integrity of the execution environment.

Linux

A term used to identify the Linux kernel, if not explicitly defined otherwise. To identify an entire operating system including the Linux kernel, the term "GNU/Linux operating system" is used.

Logical Block Addressing

A common scheme used to specify the location of blocks of data resident on computer storage devices, which are generally secondary storage systems such as hard disks.

Mandatory Access Control

Refers to a means of access control in computer security that is defined by the Trusted Computer System Evaluation Criteria (often referred to as The Orange Book by the Department of Defense) as a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e. clearance) of subjects to access information of such sensitivity.

Non-Migratable Key

Contrary to a migratable key, a non-migratable encryption key is guaranteed to reside in a TPM. A TPM can create a certificate stating that a key is a Non- Migratable Key.

Open Secure Loader

A secure boot loader developed by TU Dresden that uses AMD's SKINIT instructions.

Platform Configuration Register

The registers of a TPM that store the configuration of the software components.

Privacy Certification Authority

A Trusted Third Party (TTP) stating that an AIK is really under control of a TPM.

Root of Trust for Measurement

The Root of Trust for Measurement (known as CRTM on PC platforms) creates the integrity measurement base of a platform. The RTM is the first element in the chain of trust, mostly proceeding with bootloader and kernel, leading to the operating system and applications. On startup, the code of the RTM, which is part of the BIOS, is executed and generates measurement values covering the components of the boot process.

Source Lines Of Code

A software metric used to measure the size of a software program by counting the number of lines of text written in the source code of the program.

Static Root of Trust for Measurement

The Static Root of Trust for Measurement (S-RTM) is the static RTM. It stores measurements in PCRs 0 - 15 and 23.

Storage Root Key

An asymmetric 2048-Bit RSA key stored inside the TPM, which is used to encrypt TPM-internal data. The SRK is created by taking ownership of the TPM and resides permanently until the owner is cleared.

TCG Device Driver Library

Library of the TPM device driver.

TCG Device Driver Library Interface

Interface of the TDDL.

TCG Software Stack

The software stack specified by the TCG that is responsible for accessing and using the TPM.

TPM Device Driver

A software module in the operating system required for communication with a TPM hardware chip.

TPM Interface Specification

Specification of the hardware interface for TPMs of version 1.2.

Trusted Computing

Components and mechanisms that are compatible with, or defined by, the specifications of the TCG.

Trusted Computing Group

An industry consortium defining several specifications required to build a trusted computing platform, incl. the TPM specification, the TSS specification and the TNC specification.

Trusted Network Connect

The TNC architecture focuses on interoperability of network access control solutions and on the use of trusted computing as basis for enhancing security of those solutions. Integrity measurements are used as evidence of the security posture of the endpoint so access control solutions can evaluate the endpoint's suitability for being given access to the network.

Trusted Platform Module

A hardware device, protected against manipulation and designated for opt-in usage, providing protected capabilities and shielded locations. The TPM is a passive component and contains engines for random number generation, calculation of hash values and RSA key generation. A TPM generates and stores keys, signs or binds data to the platform and measures the platform's current state.

TSS Core Service

The TCS coordinates TPM access of multiple TSPs.

TSS Service Provider

A service dedicated to every application using TSS services in order to communicate with the TSS stack.

TSS Service Provider Interface

The interface of the TSP.

Unified Modeling Language

A standardized specification language for the modeling of objects in the context of software engineering; includes a graphical notation used to create an abstract model of a system, referred to as a UML model.

Virtual Machine

A Virtual Machine (VM) is a software implementation of a machine or a computer that behaves like a physical machine from the operating systems perspective. Virtual Machines need the presence of a software layer (VMM) in order to access the multiplexed physical hardware.

Virtual Machine Monitor

The virtual machine monitor (also called Hypervisor) generates and manages virtual machines (VM). Hardware ressources are shared to allow execution of multiple operating systems on one host.