Journal of Universal Computer Science, vol. 16, no. 4 (2010), 531-542
submitted: 10/10/09, accepted: 21/12/09, appeared: 28/2/10 © J.UCS

Towards a Virtual Trusted Platform

Martin Pirker
(Graz University of Technology
Institute for Applied Information Processing and Communications (IAIK)
Inffeldgasse 16a, 8010 Graz, Austria
martin.pirker@iaik.tugraz.at)

Ronald Toegl
(Graz University of Technology
Institute for Applied Information Processing and Communications (IAIK)
Inffeldgasse 16a, 8010 Graz, Austria
ronald.toegl@iaik.tugraz.at)

Abstract: The advances and adoption of Trusted Computing and hardware assisted virtualisation technologies in standard PC platforms promise new approaches in building a robust virtualisation platform for security sensitive software modules. The amalgam of these technologies allows an attractive off-the-shelf environment, capable of supporting security levels potentially higher than commonly deployed today. This article proposes a practical approach of combining technology elements available today to create such a platform using available components. The design supports operating high-security and low-security compartments side by side. The high security compartment is able to use the functionality of the Trusted Platform Module. The low security compartment is isolated through hardware-assisted virtualisation. The platform boots via Intel Trusted Execution Technology to resist manipulation. We discuss the building blocks of the architecture and present a number of open research challenges.
Key Words: trusted computing, virtualisation, security
Category: D.4.6

1 Introduction

The unstoppable pervasiveness of information technology applications throughout our daily lives moves security of data processing and associated risks and problems into focus. Conventionally, protection software has been employed to detect security threats such as computer viruses or trojan horses. However, security features provided by software alone can be similarily circumvented by software based attacks. The combination of recent improvements to x86 computer architectures promises a possible solution.

Virtualisation is a methodology of dividing the resources of a computer into multiple execution environments, by applying concepts such as hardware and software partitioning, time-sharing, machine simulation or emulation. Modern architectures are based on a single *hypervisor*, a control entity which directly runs on the hardware and manages it exclusively. It then manages the creation, execution and hibernation of isolated *compartments*, each hosting an unmodified operating system.

The concept of *Trusted Computing* extends the standard PC architecture with elements such as the *Trusted Platform Module* (TPM)[TCG(2007)]. The introduction of this additional hardware component allows for increased assurance that a measurement and report of the state of a system is accurate and authentic. The hard-to-tamper nature of the TPM, along with recent advances in chipsets and CPU components, allows for new approaches to build more secure and robust systems.

The advantages of *combining* virtualisation with Trusted Computing are manifold. First of all, separating the execution environment of different applications naturally results in a higher attack resistance and availability, as it allows to contain software attacks to a single compartment. Demonstrations of such systems are Terra [Garfinkel et al.(2003)], or the results of the [EMSCB(2004)] and [OpenTC(2005-9)] projects. More recently, hardware platforms have become available [Grawrock(2009)] that support the challenges of secure virtualisation by providing strong isolation of compartments on commodity hardware.

The second main advantage of virtualisation is that it helps to reduce the number of software components which influence the system state, and therefore its trustworthiness. A main difficulty [England(2008)] for the Trusted Computing concept of Attestation is keeping track of known-good system configurations. This proves extremely challenging due to the high number of possible software component combinations found in today's complex and constantly-updated operating systems. With virtualisation, the hypervisor can reduce the measurement complexity drastically by simply performing a single measurement of the complete compartment image file.

Thirdly, modern platforms also provide an alternative to the TCG model of a *static* chain-of-trust, which envisioned a continuous transition between authenticated states from platform boot onwards. They now provide the possibility of a *dynamic* switch to one special well-defined trusted system state. A new CPU instruction allows to switch the system into a secure state and then measure and run a piece of software, typically a hypervisor, which has full system control. Close hard-wired cooperation of CPU, chipset and TPM guarantees that the result is accurate. Thus, influences from boot-time only components, as for example the BIOS, can be ruled out.

In this article we propose a system architecture which integrates the dynamic switch into a trusted hardware state with an up-to-date hypervisor. Furthermore, we restrict the execution of critical compartments to trustworthy platform configurations, while at the same time describe a set of operational procedures which help us retain flexibility with respect to configuration changes. We also outline open future research challenges.

1.1 Trusted Computing

Today, Trusted Computing is commonly understood as based on the specifications of the *Trusted Computing Group*[TCG(2003)]. The TCG concepts extend current platforms with an additional hardware component, the *Trusted Platform Module* (TPM)[TCG(2007)]. Increasingly, common available PC hardware ships with an on-board TPM.

The *TPM* functions as the core trusted reporting component in the system. The TPM supports cryptographic operations and associated protocols. The support includes a hardware random number generator, RSA cryptography, SHA-1 hashing and protected non-volatile memory. One of the most important features is the set of *platform configurations registers* (PCRs). At platform reboot the PCRs are initialised to a known fixed value and then can only be changed by the PCR *extend* operation. It concatenates the current content of a PCR register with given, new data and stores the hash of the result in the same PCR. Thus, by the very nature of the SHA-1 cryptographic hash function, PCR registers attain the same content after reboot only by receiving the exact same input in the exact same order.

At power-on a platform is in a well known, freshly initialised state. The extend mechanism can then be applied to exactly document the software (code and configuration) executed in a system. All code binaries must be measured into PCRs before execution is passed on to them. Starting from the BIOS bootblock to BIOS main core, from BIOS option ROMs to the master boot record (MBR), covering bootloader, kernel, and system libraries etc., up to application code, an uninterrupted *chain-of-trust* can, in principle, be constructed. The exact configuration of the platform is mapped to unique PCR values. If a certain system state fulfills given security or policy requirements, we refer to it as a *trusted state*.

Any current state of the PCRs can be cryptographically signed by the TPM *Quote* operation with a TPM identity key, thus producing proof of the current configuration. An accompanyning measurement log documents how it was reached.

1.2 Existing Trusted Virtualisation Platforms

Several previous efforts applied the Trusted Computing chain-of-trust concept into practical application using a virtualised platform.

The [OpenTC(2005-9)] project researched, built and assembled multiple components necessary to build a demonstration operating system with integrated Trusted Computing support. The project publicly released two demonstrators [OpenTC(2009)]. The first showcases a Private Electronic Transactions (PET) scenario in which communications endpoints mutually attest their system state by using Trusted Computing technology. The second is called a Corporate Computing at Home (CCAH) scenario. In this scenario virtualised (Trusted Computing) attested compartments run side-by-side high security (corporate computing) and low security (home software) system images. The OpenTC project applied the technologies available during the project's lifetime. These includes building a chain-of-trust from the BIOS to the bootloader via Trusted Grub [Marcel Selhorst(2006)]. The system hypervisor component is provided by Xen [Barham et al.(2003)] or a L4 variant [Dresden(1999)] The OpenTC project successfully demonstrated a Trusted Computing enhanced system.

Likewise, the [EMSCB(2004)] platform demonstrates TPM-based Trusted Computing on a L4-Fiasco-based virtualisation platform. Microsoft's now defunct NGSCB[England et al.(2003)] envisioned the security critical Nexus kernel to provide an environment for security critical services, while running a legacy OS in parallel. [Coker et al.(2008)] describe a Xen-based platform which is optimized for flexible Remote Attestation.

2 Towards A Practical Platform

These previous proposals share certain characteristics. First, hardware virtualisation or isolation features are not used. Second, the chain-of-trust is established only at system boot time.

However, we believe that a chain of trust starting with the BIOS may be too brittle in practice. Consider a BIOS update, some configuration change, a new or different option ROM due to hardware change and decades of old legacy code - reaching a predictable PCR state after BIOS boot seems to be impractical. The other end of the measurement chain, the OS/application side, is also unstable. A modern operating system consists of too many components (plus component subversions), which are not loaded in strict order, thus changing the resulting PCR values. In practice it is very likely that creating a chain-of-trust fails.

Even if a reproducible (in effect constant) configuration can be established (i.e. by CD boot images), a major problem remains: Reaching a trust decision in a Remote Attestation scenario is extremely difficult due to the many possible combinations of measurements and lack of trusted known-good-value repositories.

We believe that any future security concept must go beyond just adding a TPM to the system. There are additional security benefits if CPU, chipset and TPM work in close cooperation, under the control of a fully measured and measuring system hypervisor software. We further believe that those powerful mechanisms can be applied to overcome many of the outlined challenges.

2.1 Modern Trusted Computing Hardware

Since 2005 mainstream CPUs of the major CPU manufacturers have gained the low-level capability to transparently pretend to an operating system that it is alone in the system, while in reality multiple OSe are running in parallel on the same hardware. Previously, this virtualisation required sophisticated software tricks like on-the-fly code rewriting. Nowadays, CPUs can enforce control of the most privileged instructions of an operating system by another instance, a hypervisor. The hypervisor is the component with full control of the physical hardware and enforces which hardware resources are assigned to which virtualisation compartment.

The isolation functionalities provided by the CPU alone are not sufficient to separate memory areas from one another, as I/O devices can manipulate main memory on their own using Direct Memory Access (DMA). A subset of most recent chipsets now allow to assign certain I/O devices to a specific virtualised compartment instance. Thus, previous attacks via manipulation of DMA transfers of e.g. firewire, graphics or network devices can be prevented.

2.2 Proposed Architecture

We identify two major challenges to practically establish trust in a software platform

1. identifying and measuring individual components in the chain of trust
2. deciding the trustworthiness of the resulting chain of trust.

In the following sections we present an architecture which reduces the complexity of these problems by reducing the length of the chain-of-trust from *both* ends.

Our approach uses two techniques to accomplish this: First, we utilize the features of the most recent Intel Trusted Execution Technology (TXT) enabled hardware platform to perform a *late launch* into a trusted state. TXT moves the switch to a measured initialisation state to a later point in time, thus shortening this side of the chain-of-trust, as more platform boot related components are ignored.

Second, we create an *in situ* image from a trustworthy hypervisor, system configuration and runtime environment. The application compartments to be executed in a compartment are encrypted, with the key being sealed to the state of the base system. This ensures that a configuration as set up by the operator is protected against manipulation. Only if the hypervisor and the associated runtime system are in the correct state the keys to decrypt the virtual application images are released. We extent the hypervisor and its support tools to allow the unsealing of a compartment only if the overall system is in a state defined to be trustworthy by the platform operator.

The added value of this architecture is more clearly understood with the observation that a late launched hypervisor does not have to restore the system to a state identical to before the late launch initiation. Instead we use the hypervisor to partition memory into a *secure* and an *insecure* world. The insecure side hosts a traditional operating system. The newly created secure section is strictly isolated by hardware virtualisation and I/O control, enforced by the hypervisor.

As security critical code will only be handled in isolation, Trusted Computing measurements and ultimately the chain-of-trust need only be extended into the secure side. Thus, virtualisation enables the processing of security sensitive data and code which requires a certain system state (PCRs content).

Overall, the shorter measurement chain provided by late launch, covering a small base system, to a secure compartment running trusted code ensures that the number of components is small and manageable.

3 Pieces of the Puzzle

In order to realise a dynamically launched and measured trusted platform on standard PC hardware, a set of building blocks is required. The following sections describe the individual hardware and software components and their specific role in the process.

Figure 1 illustrates the overall architecture and as well as the timeline at system start-up. At first, a conventional boot is performed by the BIOS using the Grub Bootloader to launch the TBoot tool, which configures and initiates the hardware-supported *late launch*. In this special mode the system control is passed on to a trusted and authenticated code module, which establishes a well-defined and secure system state. The module then returns control to TBoot, which subsequently launches a Linux kernel based hypervisor service. This secure and trusted platform can now initialize trusted as well as untrusted application compartments which are isolated from each other. The TPM serves as a trust anchor for measuring and reporting on the state transitions and as protected storage for system launch configurations.

In the following sections we discuss the building blocks of our architecture and of the boot process in more detail.

3.1 Intel TXT and Late Launch

Amongst a number of security improvements, a shortening of the chain-of-trust has become possible with the introduction of Intel Trusted Execution Technology[Intel Corporation(2008a)] enabled platforms. Platforms branded as

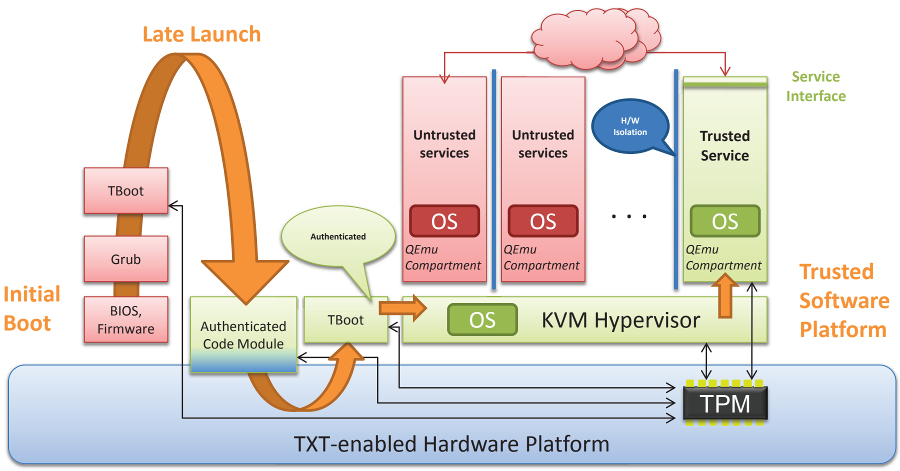

Figure 1: Overview of the software architecture and the timeline of the late launch into a measured/trusted mode. Untrusted software is shown red, measured/trusted software green and trusted hardware blue. The timeline is indicated in orange.

such possess all of the capabilities described in Section 2.1. Furthermore, they enable a *late-launch*, a dynamic switching to a measured system state at any point in time. This removes the need to maintain a (potentially very long) chain-of-trust, started at the time of platform power-on.

A late launch is initiated by the special TXT CPU instruction `SINIT`. It stops all processing cores except one, the chipset locks down all memory to prevent outside modification by devices and initialises specific TPM PCRs to their initial state. A special Intel provided (and cryptographically signed) Authenticated Code Module (ACM) initiates this chain of trust by initialising the platform to a well-defined state. Subsequently, the next software module is measured and execution control is passed to it - typically a hypervisor. Step by step the hypervisor decides which system resources to unlock and which software services to continue or to start anew.

The SINIT initialisation code is also capable of enforcing specific launch control policies (LCPs). The LCP is stored in the non-volatile storage of the platform's TPM and can only be set and modified with the authentication of the TPM owner. SINIT measures the next software component to be executed and uses the LCP to decide what to do with the measurement result. The simplest policy type *any* essentially allows anything to be executed. However, the policy can also contain a list of reference measurements. In this case the TPM owner decided that SINIT should only allow execution of one of the explicitly listed *known-good* modules, which he deems trustworthy. A software modification which was not authorized by the TPM owner is not allowed to continue.

Additional new functionalities provided by TXT aid security. For instance, the chipset is capable of automatically scrubbing system memory whenever a system reboot is initiated. This technique is also useful whenever a virtualisation instance is shut down by the hypervisor. This prevents leakage of secret data over operating system lifetime, for example.

3.2 Tboot

Trusted Boot (tboot) [Intel Corporation(2008b)] is an openly available tool to initiate a TXT late launch. Tboot is not designed to be executed inside a running operating system. Rather, it positions itself in the system boot-chain between a standard bootloader and the operating system kernel. Tboot currently supports launching Xen and alternatively the Linux kernel after a successful TXT late launch.

Typically, a Linux kernel is loaded and started by a bootloader such as GRUB [GNU(1999)], which resides in the Master Boot Record (MBR) of the harddisk. The GRUB configuration specifies the Linux kernel image to boot and additional required modules such as an initial ramdisk (initrd). For a dynamic boot this configuration order needs to be modified: the tboot binary is positioned into the first slot, before all other entries follow. The last configuration entry is the TXT specific SINIT authenticated code binary.

With this modified GRUB configuration tboot is executed first. Tboot prepares the system for TXT mode and initiates the dynamic late launch into a measured system, which includes executing the SINIT module. After successful TXT setup control is handed back to Tboot. Tboot measures the Linux kernel and passes control to it, continuing a standard Linux boot process. The Linux kernel must be Tboot aware, otherwise the memory and I/O configuration security modifications employed by SINIT and Tboot will immediately cause a security violation and TXT enforces a system reboot as a security measure.

The well-defined initialisation performed by SINIT in combination with Tboot provides the anchor of a shortened chain-of-trust. Thus, a TXT launch removes all the effects of BIOS code executing before the system bootloader and provides a robust first link for the chain-of-trust to the kernel.

3.3 KVM

A hypervisor which is able to partition the system resources into isolated compartments is provided by KVM [Qumranet(2006)] [Kivity et al.(2007)]. The recently integrated hardware virtualisation support in CPUs offloads many of the tasks needed to virtualise a full operating system to hardware. This reduces code size and complexity, and thus positively affects software security. With the KVM hypervisor architecture the core, the small and critical virtualisation component, is a loadable module of a standard Linux kernel.

The KVM kernel module monitores the low-level machine operations of the virtualized systems. Whenever execution of privileged operations or I/O access is attempted within the compartment executed in guest mode, KVM takes control and decides on how to proceed. Privileged Instructions are emulated by KVM and I/O is forwarded to a QEMU [Bellard(2005)] support process, which runs with standard Linux user process privileges.

The KVM approach provides several benefits to our architecture. First, KVM is integrated into the standard Linux kernel tree and therefore always in sync with ongoing kernel developments. This affects security fixes as well as the ability to immediately take advantage of Linux kernel improvements to perform on up-to-date platforms. Furthermore, virtualisation always requires support components to provide device emulation and enable communication. In the KVM model this is handled by an associated QEMU user process, which in turn uses the services of a normal Linux (kernel) runtime.

In case one QEMU instances crashes, just like a normal misbehaving user process, the offending one is removed - all other virtualisation are not affected.

It is also possible to control the code size of the QEMU executable. It can be compiled to only contain the absolutely necessary functions for a specific task. This allows run multiple virtualisations with different requirements in parallel. From a security perspective, a small size of critical code provides a better starting point for intensive scrutinization of the code in security reviews.

3.4 A Lightweight Linux Base System

Of course virtualisation with KVM requires a basic Linux system as support. To integrate it in the measurement process, we slightly modify the Linux boot process by integrating all pieces integrated into one large binary image consisting of the Linux kernel, the initial ramdisk and the base system. This binary is specified in the GRUB configuration as kernel and thus automatically measured by tboot. Thus, any modification of the base system is reflected in different TPM PCR values.

The size of the Linux base system depends on the requirements of the components which it needs to run or support. The system must provide all the capabilities so that QEMU can provide the I/O device emulations needed for all KVM virtualised compartments. For example, if all KVM guests are e.g. server services which do not require audio support QEMU can be compiled without sound device emulation and the base system can be cleaned of audio components, too.

Additionally, there is a need for components to manage the virtualised applications themselves. In the simplest case this is an SSH attached service which allows login, transfer and exchange of application images.

All virtualised application images are encrypted with AES. KVM is capable of en-/decrypting virtualisation images on-the-fly if provided with the proper cryptographic key. We seal this key to a trusted PCR state. Thus, whenever a trustworthy base system (as defined by the TPM owner) is booted, the AES key can be unsealed by a startup script and provided to KVM to run an encrypted application image. If the base system were, possibly maliciously, modified, this will fail.

4 Discussion

Combining the components discussed in Section 3 allows to assemble a system as proposed in Section 2.2. The architecture allows for different approaches in solving certain problems and some issues need further research to find a solution of manageable complexity.

4.1 KVM and QEMU security

The security of QEMU device emulation is one of the critical issues. A failure in the emulation potentially allows a “jail-break” of a virtualisation guest and consequently access of the host system. As any other piece of software, the QEMU software is not error-free [Debian(2009)]. However, the Unix user vs. root security privilege model also applies for the QEMU userspace process [Kivity(2008)] and it is therefore possible to run each virtualisation instance under a distinct user ID. For higher security scenarios advanced mandatory access control system such as [NSA(2000)] provide a hardened attack protection.

4.2 System Maintenance

The system requires an initial setup procedure. In this step the TPM ownership is taken, the hard disk is partitioned, formatted, the boot loader is installed and the necessary binaries are copied to the disc. The TPM non-volatile storage is configured with the TXT launch control policy containing the correct hash of the installed binaries.

As outlined in Section 3.4, the base system is assembled into one large binary. As a direct consequence upgrading the system becomes a complex task. Any change in the configuration or binaries causes a new known-good value, which has to be updated in the LCP stored in the TPM. Furthermore, all keys for the virtualised compartments need to be resealed to the new state.

4.3 TPM multiplexing

The architecture outlined in Section 2.2 proposes a separation into a secure and an insecure world. The chain-of-trust extends only into one compartment of the secure world. First, the base system uses the TPM to establish a trusted system state for the compartment and then starts it. Now the base system can pass on the responsibility for the TPM to the appication compartment by granting it exclusive access to the hardware device. This can be achieved by adding a TPM device emulation to QEMU [Bleher(2007)], which appears to the operating system inside just like a hardware TPM. QEMU forwards the byte streams to the `/dev/tpm` interface in the base system, to which the hardware TPM is connected.

This solution only allows one virtualised compartment to run Trusted Computing applications. Multiple approaches have been proposed to solve the multiplexing of the TPM, for example virtual software TPMs [Berger et al.(2006), Scarlata et al.(2007)].

5 Conclusions

In this paper we propose a way to bring Trusted Computing technologies closer to practical deployment. We describe how currently available hard- and software components can be combined on the PC platform to create a system capable of running multiple trustworthy compartments in parallel. By constructing a short chain-of-trust using late launch via a hypervisor to virtual compartments, we reduce complexity to manageable levels. We are currently in the process of implementing a platform prototype. Using Intel TXT technology and sealing virtualisation compartments to a specific system state helps to guarantee and enforce the software integrity of the platform so that it resists malicious modifications.

Acknowledgments

This work was supported by the Österreichische Forschungsförderungsgesellschaft (FFG) through project acTvSM, funding theme FIT-IT, no. 820848.

References

[Barham et al.(2003)] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: "Xen and the art of virtualization"; SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles; 164–177; ACM, New York, NY, USA, 2003.

[Bellard(2005)] Bellard, F.: "Qemu, a fast and portable dynamic translator"; ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference; 41–41; USENIX Association, Berkeley, CA, USA, 2005.

[Berger et al.(2006)] Berger, S., Cáceres, R., Goldman, K. A., Perez, R., Sailer, R., van Doorn, L.: "vTPM: virtualizing the trusted platform module"; USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium; 305–320; 2006.

[Bleher(2007)] Bleher, T.: "Atmel TPM support for QEMU"; http://www.mail-archive.com/qemu-devel@nongnu.org/msg13408.html (2007).

[Coker et al.(2008)] Coker, G., Guttman, J., Loscocco, P., Sheehy, J., Sniffen, B.: "Attestation: Evidence and trust"; (2008).

[Debian(2009)] Debian: "QEMU Debian security advisory 1799 (may 2009)"; http://www.debian.org/security/2009/dsa-1799 (2009).

[Dresden(1999)] Dresden, T.: "The L4 -kernel family"; http://os.inf.tu-dresden.de/L4/ (1999).

[EMSCB(2004)] EMSCB: "The European Multilaterally Secure Computing Base (EMSCB) project"; http://www.emscb.org/ (2004).

[England(2008)] England, P.: "Practical techniques for operating system attestation"; (2008).

[England et al.(2003)] England, P., Lampson, B., Manferdelli, J., Willman, B.: "A trusted open platform"; Computer; 36 (2003), 7, 55–62.

[Garfinkel et al.(2003)] Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: "Terra: A virtual machine-based platform for trusted computing"; Proceedings of the 19th Symposium on Operating System Principles(SOSP 2003); 193–206; ACM New York, NY, USA, 2003.

[GNU(1999)] GNU: "Gnu grub bootloader"; http://www.gnu.org/software/grub/ (1999).

[Grawrock(2009)] Grawrock, D.: Dynamics of a Trusted Platform: A Building Block Approach; ISBN 978-1934053171; Richard Bowles, Intel Press, Intel Corporation, 2111 NE 25th Avenue, JF3-330, Hillsboro, OR 97124-5961, 2009.

[Intel Corporation(2008a)] Intel Corporation: "Intel Trusted Execution Technology Software Development Guide"; http://download.intel.com/technology/security/downloads/315168.pdf (2008a).

[Intel Corporation(2008b)] Intel Corporation: "Trusted boot (tboot)"; http://sourceforge.net/projects/tboot/ (2008b).

[Kivity(2008)] Kivity, A.: "How KVM does security"; http://avikivity.blogspot.com/2008/05/how-kvm-does-security.html (2008).

[Kivity et al.(2007)] Kivity, A., Kamay, V., Laor, D., Lublin, U., Liguori, A.: "kvm: the Linux Virtual Machine Monitor"; OLS2007: Proceedings of the Linux Symposium; 225–230; 2007.

[Marcel Selhorst(2006)] Marcel Selhorst, C. S.: "Trustedgrub"; http://sourceforge.net/projects/trustedgrub/ (2006).

[NSA(2000)] NSA: "Selinux - security-enhanced linux"; http://www.nsa.gov/research/selinux/index.shtml (2000).

[OpenTC(2005-9)] OpenTC: "The Open Trusted Computing (OpenTC) project"; http://www.opentc.net/ (2005-9).

[OpenTC(2009)] OpenTC: "OpenTC project demonstrators CD images"; ftp://ftp.suse.com/pub/projects/opentc/ (2009).

[Qumranet(2006)] Qumranet: "KVM - Kernel-based Virtualization Machine"; http://www.qumranet.com/files/white_papers/KVM_Whitepaper.pdf (2006).

[Scarlata et al.(2007)] Scarlata, V., Rozas, C., Wiseman, M., Grawrock, D., Vishik, C.: "Tpm virtualization: Building a general framework"; N. Pohlmann, H. Reimer, eds., Trusted Computing; 43–56; Vieweg, 2007.

[TCG(2003)] TCG: "Trusted computing group website"; http://www.trustedcomputinggroup.org/ (2003).

[TCG(2007)] TCG: "TPM specification version 1.2 revision 103"; https://www.trustedcomputinggroup.org/specs/TPM/ (2007).