A Practical Approach for Establishing Trust Relationships between Remote Platforms using Trusted Computing

Kurt Dietrich, Martin Pirker, Tobias Vejda, Ronald Toegl, Thomas Winkler and Peter Lipp

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A–8010 Graz, Austria
{kdietrich,mpirker,tvejda,rtoegl,plipp}@iaik.tugraz.at
{thomas.winkler}@uni-klu.ac.at

Abstract. Over the past years, many different approaches and concepts in order to increase computer security have been presented. One of the most promising of these concepts is *Trusted Computing* which offers various services and functionalities like reporting and verifying the integrity and the configuration of a platform (attestation). The idea of reporting a platform's state and configuration to a challenger opens new and innovative ways of establishing trust relationships between entities. However, common applications are not aware of Trusted Computing facilities and are therefore not able to utilise Trusted Computing services at the moment. Hence, this article proposes an architecture that enables arbitrary applications to perform remote platform attestation, allowing them to establish trust based on their current configuration. The architecture's components discussed in this article are also essential parts of the OpenTC proof-of-concept prototype. It demonstrates applications and techniques of the Trusted Computing Group's proposed attestation mechanism in the area of personal electronic transactions.

1 Introduction

Trusted Computing (TC) is constantly gaining ground in industry and the public perception of Trusted Computing is starting to improve [6]. A central role is played by the Trusted Computing Group (TCG) [18] which is specifying the core components, namely the Trusted Platform Modules (TPM) and surrounding software architectures like the TCG Software Stack (TSS) [15]. Based on these components, security and trust related services like *remote attestation*, *sealing* or *binding* are defined.

Hence, in the first contribution the question how trust relationships between remote platforms can be established by using TC is addressed. The approach presented in this paper allows to establish trusted communication channels by means of the TCG's specified remote attestation. The approach introduces a so-called *attestation proxy* that is placed in front of the actual application and performs a mutual platform attestation of the two communication parties. The

actual communication channel is only established if the attestation succeeded. This approach allows legacy applications to benefit from attested communication channels without the need to modify the application code.

As the proof-of-concept implementation is done in Java, the second contribution deals with the problem how TC concepts can be integrated into virtual machine based runtime environments such as Java™. Questions to be answered are how to measure loaded class and jar files, how to deal with external resources or how to handle calls to native code.

The basis for all TC related services is the TPM. The TPM is a hardware chip providing essential functionality for a TC enabled system like a RSA engine, a true random number generator or mechanisms to securely store and report the state of a system. While TPMs are produced and shipped by a variety of manufacturers, important software components like the trusted software stack are not widely available yet. The presented IAIK TSS for the Java Platform (jTSS [14]) provides TC services to applications and manages the communication with the TPM. The jTSS provides the foundations for the two main contributions of this work.

1.1 Related Work

The idea of remote attestation has been pursued by various research groups. Hence, many different approaches discussing this research area have been published. The most important are introduced in the following paragraphs.

The concept of *Property-Based Attestation* (PBA) [11] provides an alternative to the attestation mechanisms specified by the TCG henceforth called *binary attestation*. A Trusted Third Party (TTP) translates the actual system configuration into a set of properties and issues certificates for those properties. During the attestation process a (remote) verifier can decide whether or not the platform security properties meet the requirements of the respective use case. In literature, using TTPs for certification of properties is called *delegation*. This scenario avoids several (undesired) drawbacks of binary attestation. For instance, presenting the concrete system configuration to a verifier is not desirable from a privacy perspective and management of all possible configurations is a difficult task.

Alternatively, *Semantic Remote Attestation* (SRA) [12] uses language-based techniques to attest high level properties of an application. The proposal is based on the Java Virtual Machine (JVM) environment which is attested by binary attestation itself. The JVM can enforce a security policy on the running code based on data flow control and taint propagation mechanisms. Hence, this approach is a hybrid approach between binary attestation and attesting properties.

Moreover, the Trusted Computing Group - as the leading group for TC specifications - has published a concept for trusted network access also known as Trusted Network Connect (TNC) [22]. TNC enforces a policy based access and integrity control by measuring the state and configuration of a platform according to specified policies. Furthermore, TNC introduces the concept of isolation. Platforms that cannot be attested correctly are *isolated*. This means that they

are not allowed to access the network unless they can successfully report that their integrity has been restored (remediation). The usage of a TPM is optional in order to make this technology available on a variety of platforms. Nevertheless, if a TPM is present it is used for extended integrity checking and binding of access credentials to the platform.

Other approaches focus on improving established protocols like SSL. The main problem these approaches deal with is that there is no linkage between the attestation information (i.e. the signed quote and the AIK certificate) and the SSL authentication information. Stumpf *et al.* [21] discuss a concept for a robust integrity reporting protocol by combining it with a key agreement protocol. The same problem is addressed by Sailer *et al.* [20]. In their paper a solution for linking SSL tunnel endpoints to attestation information by adding the SSL public key to the event log and PCRs is discussed. Furthermore, they introduce a new certificate type, the so-called platform property certificate that links an AIK to a SSL private key. Binding the keys with the certificate should prevent the misuse of a compromised SSL key.

1.2 Outline of the Paper

The remainder of this paper is organised as follows: Section 2 gives details about the overall architecture. Section 2.1 describes the attestation proxy and illustrates the use of the concept of remote attestation to establish and validate trusted relationships between two entities. Section 2.2 presents an outline of the IAIK jTSS, discussing the overall structure as well as implementation concepts. Section 2.3 deals with aspects of adapting the Java virtual machine to be fully integrated into TC environments. Section 2.4 explains the link between TPM based keys and public key infrastructure concepts. Finally, Section 3 concludes the paper.

2 The Proof of Concept Architecture

In this section, the overall architecture and actions of the proposed approach are briefly discussed. As shown in Figure 1 the architecture includes a proxy that provides an attestation service to applications, a trusted software stack (jTSS) and the trusted Java VM. The integrity of all components of the architecture is measured as defined by [24] in order to establish a chain-of-trust starting from the platform's BIOS up to the proxy service (see Figure 1). However, in order to build the full chain, the architecture requires further components. These components include a core-root-of-trust for measurement (CRTM)[1] that is included in the BIOS, a trusted boot loader (e.g. Trusted Grub [19]) and a trusted operating system. They are out of scope for this implementation and are therefore not discussed in this paper. Nevertheless, the architecture assumes that the platform performs an authenticated boot as defined by [18].

[1] Modern computer systems use a dynamic-root-of-trust for measurement (DRTM)

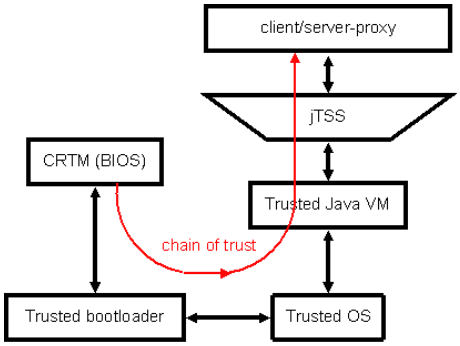

Fig. 1. Attestation Service Architecture and *Chain-of-Trust.*

The scenario depicted in Figure 2 is as follows: A client application wants to establish a connection to a server based service. The application could be a web-browser or any application that requires a network connection[2]. However, the application is only allowed to connect if the trust state of the client and the server meet specified requirements that are defined by policies. The trust state in the context of TC is derived from the software components that are running on a platform and the hardware the platform is equipped with. Consequently, the policies include certain sets of allowed hardware and software configurations.

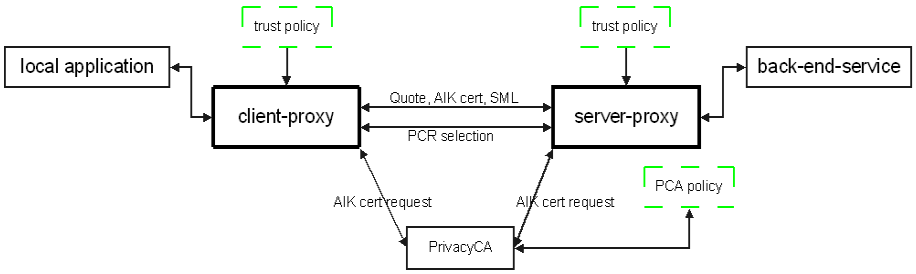

Fig. 2. Attestation Process Scenario

The platform state has to be reported to the remote platform which is then verified by it. To allow this, the presented architecture is embedded in the context of a trusted computing enhanced Public Key Infrastructure (PKI).

Each component of the proposed approach is discussed in detail in the following sub-sections.

[2] The scenario focused on within OpenTC uses a web browser as application and a bank server as back-end service, nevertheless the architecture can be used with any arbitrary application and service.

2.1 The Attestation Proxy

The attestation-proxies are responsible for attesting the platforms and routing the network traffic between the platforms, the client application and the back-end service (Figure 2). Additionally, the proxies exchange measurement- and attestation values of the platforms. Consequently, they are also responsible for validating the measurement values according to preset policies.

The platform attestation with the proxy is as follows: the client-proxy receives a connection request from a local application and opens a channel to the server-proxy. Prior to forwarding the data received from the application, the proxy initiates the attestation sequence. This sequence includes the following steps:

Depending on the proxy policy, the proxy may use a previously generated attested identity key (AIK) or may create a new one. Reusing of the AIK from a previous proxy connection saves the time for creating a new one. However this potentially lowers the level of privacy. When a new identity key is created in the TPM, the key has to be attested by a Privacy CA which issues a corresponding AIK certificate. The key is then used to sign the content of the PCR register.

The state of the system is reflected in the Platform Configuration Registers (PCR) of a TPM. The client-proxy proves to the server-proxy that the system is running in a desired trusted configuration by running the special TPM "quote" operation. It reports the content of a selected set of PCR registers and signs this information with an identity key.

As shown in Figure 2 the proxy sends the following items: the quote blob, the AIK certificate and the Stored Measurement Log (SML). The verification component of the proxy is now able to determine the state of the remote platform by evaluating the quote blob and the SML. The SML contains a list of all software components that have been loaded on the remote platform including their hashes. By recalculating the hashes and comparing them with the hash from the quote blob, the proxy has evidence of the remote platforms state. Furthermore, the signature on the quote blob is verified with the help of the included AIK certificate. If required, the Privacy CA is contacted for additional data (i.e. CRLs, OCSP requests) for verification of the certificate itself.

Only after all attestation steps have been successfully completed and both platforms have validated and accepted each other's state and configuration the connection between the application and the back-end service is permitted.

The attestation process in the depicted scenario is done in both directions. Other scenarios might require only the server or the client to be attested.

In order to access TC services and the TPM, the proxy relies on the trusted Java stack. It provides TC services to application like the proxy and manages the communication with the TPM. The trusted Java stack is discussed in the following section.

2.2 The Trusted Software Stack

The TCG not only specifies TPM hardware but also defines an accompanying software infrastructure called the TCG Software Stack (TSS) [15]. The stack

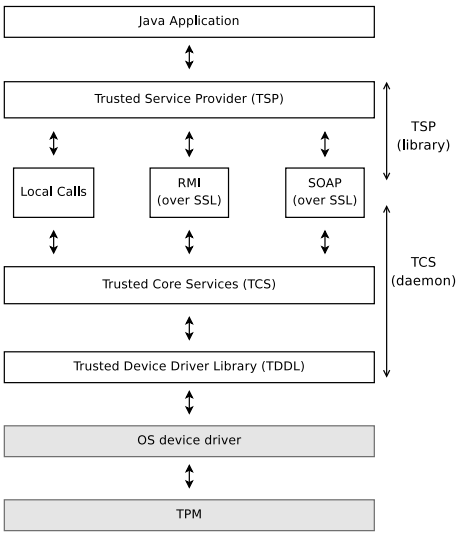

Fig. 3. Overview of jTSS Stack Layered Architecture

consists of different modules: the Trusted Service Provider, the Trusted Core Services and the Trusted Device Driver Library. The exact requirements of these modules can be found in [15]. A Java specific discussion is provided in the next sections.

Architecture The TCG chose a layered architecture, which specifies interfaces in the C programming language, thus allowing applications to access the Trusted Computing functionality in a standard way. At the time of writing, several implementations for specific operating systems are available [2] or under development [5]. Up to now, aside from the TrouSerS TSS stack [9], the here presented IAIK TSS for the Java Platform (jTSS) is the only TCG software stack available as open source. All other known implementations are proprietary meaning that they only support TPMs from specific manufacturers .

The architecture presented in this paper allows operating system independence by providing the TC functionality within the Java programming language. At the same time, different TPM implementations, including a software based emulator, are supported. Thus actual platform-independent trusted services can be built on top of the presented TCG Software Stack for the Java Platform (jTSS). In contrast to other projects [7] that implement only sub-sets of the functionality, this stack closely follows the specification and includes both, high and low level APIs as proposed by the TCG. The different layers of the stack architecture are presented in Figure 3 and discussed in the following paragraphs.

The application level Trusted Service Provider Java applications can access Trusted Computing functionality by using a derivate of the Trusted Service

Provider (TSP) interface. By providing an object oriented interface, the application developer is relieved from internal handle and memory management. A context object serves as entry point to all other functionality such as TPM specific commands, policy and key handling, data hashing and encryption and PCR composition. In addition, command authorisation and validation is provided and user owned cryptographic keys can be held in a per-user persistent storage.

Each application has an instance of the TSP library running on its own. This TSP communicates with the underlying Trusted Core Services (TCS). Different means of communication are possible. For small set-ups and for testing, a local binding using standard java function calls is used. However, the TCS may also run on another machine. In this case, Java Remote Method Invocation (RMI) may be used. Here, the communication between the two modules can be protected with Secure Socket Layers (SSL).

In addition to this implementation specific interface, the TSS standard also calls for an alternative interface utilising the Simple Object Access Protocol (SOAP) [16], which is implementation and platform independent.

The Trusted Core Services The Trusted Core Services (TCS) are implemented as a system service, with a single instance for a TPM. By ensuring proper synchronisation, it is designed to handle requests from multiple TSPs. Among the main functionalities implemented in the TCS are key management, key cache management, TPM command generation and communication mechanisms.

Since the hardware resources of the TPM are limited, the loading, eviction and swapping of keys and authorisation sessions needs to be managed. Keys can be permanently stored in and retrieved from the system persistent storage using globally unique UUIDs [13]. With these mechanisms, complex key hierarchies can be defined, allowing to implement domain (i.e. enterprise) wide policies. The TCS event manager handles the SML, where PCR extension operations are tracked. For low level access, commands and data are assembled.

Low Level Integration The TCS communicate with the TPM via the TSS Device Driver Library (TDDL). For hardware access, the Java objects need to be mapped to the standardized C-structures. Primitive data types need to be converted as well, considering the byte order of the host platform. These structures are then processed as byte streams. Since all commands and data are sent as such plain byte streams, this allows for an OS and hardware-independent implementation.

In the Linux operating system, hardware-vendor specific driver modules and a generic driver are integrated in recent kernel releases. The TPM can be accessed through the */dev/tpm* device. With Microsoft Windows Vista, a generic system driver for version 1.2 TPMs is supplied. With the so called Trusted Base Services (TBS) [8], basic functionality like resetting or taking ownership is provided and TSS implementations can be supported. To integrate this Windows interface in the Java environment, a small native C helper library is accessed via the Java

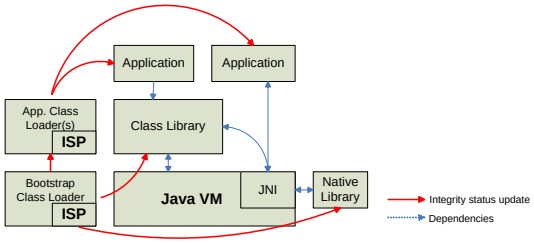

Fig. 4. Transitive trustmodel for the Java VM.

Native Interface (JNI). The already pre-assembled byte stream is passed on as a command to the TPM via the TBS, and the response stream is returned to the Java interface.

2.3 The Trusted Java Virtual Machine

All currently proposed attestation mechanisms rely on integrity measurement of the software stack running on a platform. This holds also true for all forms of property-based attestation. In our work, we extended the trust chain to the Java VM as shown in Figure 4. We describe the additions to the Java VM in this section starting with trusted class loading.

Trusted Class Loading Dynamic class loading is a feature of the Java VM specification. Classes are loaded during run time from any location pointed to by the class path. The class loaders form a tree structure to enable a delegation model. A class loader can delegate the loading of classes to a parent classloader and, if the loading fails, try to locate and load classes for itself. The root of the tree is the so-called bootstrap (or primordial) class loader. The loaded classes are assigned to so-called *protection domains* which prevent leakage of information between trusted code and application-specific code. Note that the term *trusted code* in this case merely refers to the class library shipped with the Java VM. However, this separation between different classes can be exploited for the functionality of our Trusted Java VM as well. The security mechanisms of the Java VM rely on the activation of the *security manager* which is enabled by default in our implementation.

The proposed approach extends class loading by measurement of executable contents which is, in the case of the Java environment, restricted to class files. Before the actual bytecode is present in the VM, the files are hashed and a PCR is extended. For the case of a secure boot functionality, the VM has the ability to terminate execution if a class file is not previously known. A special case for class loading is the reflection API of the Java language. Using qualified names, the application designer can dynamically load classes. For trusted class loading, this has no impact as those classes are loaded through the usual class loading mechanism and are measured as well.

JAR-files contain a collection of class files and their measurement offers the possibility to reduce the *PCR extend* calls to the TPM. Our experiments with the measurement architecture show that measurement of single class files can significantly affect the performance of class loading if the number of class files of the application grows large. If JAR-files are measured on the other hand, this overhead can be reduced to a minimum. As JAR-files are a usual way to distribute Java applications, this approach is the most practical one.

Other files that affect the security of Java applications and the Java VM itself are configuration files such as the Java security policy. For measurement, configuration files (and hence the subsequent configuration) are equal if and only if their hashes are equal. However, innumerous possibilities of formatting leading to the same configuration exist. As this provides no robust means to determine security, we decided to skip their measurement altogether.

Java Native Interface The Java Native Interface (JNI) allows the application designer to use programming languages such as C/C++ or assembly to interact with Java applications. The interface is two-way, which means that the native code can also access Java objects, i.e. create, inspect and update them, call Java methods, catch and throw exceptions, load classes and obtain class information. Whereas there are applications where this proves to be useful, from a security perspective native libraries pose potential threats.

IBM designed an integrity measurement architecture on a Linux environment [1]. In their design, they intercept a set of system calls where files (executables, libraries, etc.) are loaded and measured into a PCR. Hence, as the VM loads the libraries dynamically, this measurement architecture would take care of the measurement and we can omit further discussion of this issue.

An alternative view on the problem is taken from an application perspective. A native library is part of the application it is used by. Hence, despite some restrictions, it might still be useful to include loaded libraries in the measurement. If there is also a measurement hook on OS level, one has to take care that the measurement is not taken twice. The general problem with this approach lies in the fact that loading of shared libraries on a Linux/GNU like environment can be followed by loading further shared libraries which is taken care of by the operating system. From the perspective of the VM, these libraries cannot be measured.

Components In this section we give a component level description of our Java VM design. To keep the design simple and the code changes to the class loader as small as possible, we chose to implement a single interface for interaction with the measurement architecture which is called *Integrity Service Provider* (ISP). It manages the integrity measurement and provides methods necessary to enforce the integrity measurement policy. The *Measurement Agent* (MA) offers an interface to measure data that is crucial for the state of the platform. For the Java VM this would be class- and JAR-files. The *Verification Agent* (VA) performs the task of verifying the measurements taken by the MA against reference

values. The *Storage Manager* class abstracts operations necessary to load and store Reference Integrity Measurements (RIMs) from a location.

In general, the storage of RIMs is a non-trivial task. Two possibilities have been proposed so far: storage inside a shielded location and the usage of RIM certificates. If the Java VM running on a device is using only a restricted number of applications the storage inside a shielded location is possible. On a general purpose computer the number of RIMs may become large which could introduce storage problems. A more practical solution would be to use cryptographic means to ensure the integrity and authenticity of RIMs which then can be kept on any type of storage [17].

Usage Model for PCRs The Integrity Measurement Architecture (IMA) proposed by IBM [1] is attached to the Linux kernel. If we compare it to our Trusted Java VM the operating system has more power to manage measurements. Obviously, the operating system never gets unloaded and hence the data structures introduced in IMA can hold links to already measured files. If a file is opened a second time, IMA hashes it and compares this hash to the value in its data structures. If the hashes are equal, everything is fine and no PCR is extended. If the hashes differ, the number of PCR is extended with the new hash value. This allows IMA to only report a file twice if it is really necessary and changes (malicious or not) of files are detected.

This mechanism cannot be adapted to the VM measurement architecture for the obvious reason that, if the VM terminates, the data structures get reset and the measurement history is no longer available.

Those facts impose several restrictions on the architecture. At first, there need to be separate registers for extending the virtual machine itself, and the applications that run on this VM. Otherwise it will not be possible to seal any Java application to this VM configuration. If we suppose the operating system takes care of the measurement of the VM, it can also detect changes in the executable and core libraries of the VM as outlined in the IMA approach. Furthermore, as files are usually not measured twice, the value in the PCR for the VM represents a unique value to which applications can be sealed to.

2.4 Trusted PKI Support

The proposed approach strongly relies on a public key infrastructure. Hence, this section discusses the components required for establishing a trusted PKI.

A trusted PKI or *trusted computing enhanced public key infrastructure* is a framework enabling authentication, confidentiality and integrity services by using public key cryptography with support of trusted computing technology. It assists entities of (public) networks to establish levels of trust and/or secure communication channels. In the following two paragraphs we describe the trust enabling components required for our architecture.

Attested (Trusted) Identity The TPM Endorsement Key (EK) uniquely identifies a TPM and hence a specific platform. Therefore, the privacy of a user is at risk if the EK would be used directly for transactions. As a countermeasure, the TCG introduced Attestation Identity Keys (AIKs) and associated AIK certificates (standard X.509 Public Key Certificates that include private extensions defined by TCG[17]), which cannot be backtracked directly to a specific platform. Still, they contain sufficient proof that the Trusted Computing supported hardware is hosting the certified key.

A trusted identity comprises two data objects: a non-migratable identity keypair hosted by a TPM and an associated certificate proving that the keypair belongs to a valid TPM, vouched for by a Privacy CA entity.

An identity key can only be used to operate upon data created by the TPM itself and not for signing arbitrary data.

Privacy CA As depicted in Figure 2, the certification of AIKs is done by a dedicated and trusted third party, the so-called *Privacy CA* (PCA). A PCA is a CA with the requirement of hiding the platform specific EK credential. In order to obtain an AIK certificate, a specific protocol between trusted platform and Privacy CA takes place: The TPM creates a request package containing identity public key, AIK certificate label and platform specific certificates. The Privacy CA checks the included information and if all pieces conform to the CA policy, an AIK certificate is issued. The response is encrypted so that only the TPM indicated in the request can extract the AIK certificate.

The mode of operation of a Privacy CA is regulated by policy. It clearly describes how the relationship between EK certificates and the issued AIK certificates is managed. The policy options for a Privacy CA cover the spectrum from "remember everything" to "know enough for the specific operation, forget everything after completion of operation". Thus, the usage of a specific Privacy CA is scenario dependent and has to consider the intended level of privacy. In a restricted deployment scenario the Privacy CA - as a central authority - issues and validates AIK certificates only from well-known clients. This requires an initial registration step of each client's EK certificate.

3 Conclusion

This paper proposes an architecture for enhancing arbitrary applications with Trusted Computing functionality. With this architecture, legacy applications can now benefit from Trusted Computing services - in this special scenario from remote attestation - without being modified. Furthermore, they are now able to derive a trust state based on the remote platforms software configuration. In order to demonstrate the feasibility of the approach a proof-of-concept prototype has been developed by implementing the architecture.

Moreover, by adapting a Java Virtual Machine, we showed that it is possible to create a chain-of-trust starting from the BIOS up to a virtualised execution environment like Java. The adapted Java VM allows user applications to execute

in a trusted environment. By integrating measurement mechanisms directly into the run time environment, high flexibility for these applications can be maintained, even within tight security requirements when building a trustworthy system.

Acknowledgements The efforts at IAIK to integrate TC technology into the Java programming language are part of OpenTC project funded by the EU as part of FP-6, contract no. 027635. The projects aims at providing an open source TC framework with a special focus on the Linux operating system platform. Started as an open source project, the results can be inspected by everybody thus adding towards the trustworthiness of Trusted Computing solutions.

References

1. R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proceedings of the 13th USENIX Security Symposium*, 223–238, 2004.
2. NTRU Cryptosystems, Inc. NTRU Core TCG Software Stack (CTSS), `http://www.ntru.com/products/tcg_ss.htm`, 2005.
3. R. Stallman. Can You Trust Your Computer? `http://www.gnu.org/philosophy/can-you-trust.html`, 2007.
4. B. Schneier. Who Owns Your Computer? `http://www.schneier.com/blog/archives/2006/05/who_owns_your_c.html`, 2007.
5. K.-Y. Baek, W. Ingersoll, and S. A. Rotondo. OpenSolaris Project: Trusted Platform Module Support. `http://www.opensolaris.org/os/project/tpm/`, 2007.
6. R. L. Kay. Trusted Computing is Real and it's Here, `https://www.trustedcomputinggroup.org/news/Industry_Data/Endpoint_Technologies_Associates_TCG_report_Jan_29_2007.pdf`, 2007
7. L. Sarmenta, J. Rhodes, and T. Müller. TPM/J Java-based API for the Trusted Platform Module. `http://projects.csail.mit.edu/tc/tpmj/`, 2007.
8. Microsoft Developer Network. TPM Base Services. `http://msdn2.microsoft.com/en-us/library/aa446796.aspx`, 2007.
9. TrouSerS - An Open-Source TCG Software Stack Implementation. `http://trousers.sourceforge.net/`, 2007.
10. S. Kinney. *Trusted Platform Module Basics: Using TPM in Embedded Systems.* Elsevier, Burlington, MA, USA, ISBN 13: 978-0-7506-7960-2, 2006.
11. A.-R. Sadeghi, and C. Stüble. Property-based Attestation for Computing Platforms: Caring about Policies, not Mechanisms. In *Proceedings of the New Security Paradigm Workshop (NSPW)*, 67–77, 2004.
12. V. Haldar, D. Chandra, and M. Franz. Semantic Remote Attestation - Virtual Machine Directed Approach to Trusted Computing. In *Proceedings of the 3rd Virtual Machine Research and Technology Symposium*, 29–41, 2004.
13. International Telecommunication Union. Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 object identifier components. ITU-T X.667. `http://www.itu.int/ITU-T/studygroups/com17/oid/X.667-E.pdf`, 2004.
14. M. Pirker, T. Winkler, and R. Toegl, Trusted Computing for the Java™Platform, `http://trustedjava.sourceforge.net/`, 2007.

15. Trusted Computing Group. TCG Software Stack Specification, Version 1.2 Errata A. https://www.trustedcomputinggroup.org/specs/TSS/, 2007.
16. W3C. Simple Object Access Protocol (SOAP) 1.1, W3C Note. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, 2000.
17. Trusted Computing Group. TCG Infrastructure Specifications. https://www.trustedcomputinggroup.org/specs/IWG, 2007.
18. Trusted Computing Group. https://www.trustedcomputinggroup.org, 2007.
19. M. Selhost, and C. Stüble. TrustedGRUB, Version 1.1, http://sourceforge.net/projects/trustedgrub, 2007.
20. K. Goldman, R. Perez, and R. Sailer. Linking remote attestation to secure tunnel endpoints. In *Proceedings of the first ACM workshop on Scalable Trusted Computing*, 21–24, 2006.
21. F. Stumpf, O. Tafreschi, P. Röder and C. Eckert. A Robust Integrity Reporting Protocol for Remote Attestation. In *Second Workshop on Advances in Trusted Computing (WATC '06 Fall)*, 2006
22. Trusted Computing Group. Trusted Network Connect (TNC) Specifications. https://www.trustedcomputinggroup.org/specs/TNC/, 2007.
23. Frederic Stumpf, Omid Tafreschi, Patrick Röder, and Claudia Eckert. A robust Integrity Reporting Protocol for Remote Attestation. Second Workshop on Advances in Trusted Computing (WATC '06 Fall), Tokyo, December, 2006.
24. Trusted Computing Group. TCG Specification Architecture Overview, Revision 1.4, https://www.trustedcomputinggroup.org/groups/TCG_1_4_Architecture_Overview.pdf, 2007.