Towards Trust Services for Language-Based Virtual Machines for Grid Computing

Tobias Vejda, Ronald Toegl, Martin Pirker, Thomas Winkler

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

Abstract. The concept of Trusted Computing (TC) promises a new approach to improve the security of computer systems. The core functionality, based on a hardware component known as Trusted Platform Module (TPM), is integrated into commonly available hardware. Still, only limited software support exists, especially in the context of grid computing. This paper discusses why platform independent virtual machines (VM) with their inherent security features are an ideal environment for trusted applications and services. Based on different TC architectures building a chain-of-trust, a VM can be executed in a secure way. This chain-of-trust can be extended at run-time by considering the identity of the application code and by deriving attestable properties from the VMs configuration. An interface to provide applications with TC services like sealing or remote attestation regardless of the underlying host architecture is discussed.

1 Introduction

Grid computing promises to provide massive computational power by distributing the workload over a large pool of independent systems. Virtual Machines (VM) allow one to overcome many of the complexities and security issues involved and are a well-suited basis. Still, open security issues exist. We use this context as example to show how these can be mitigated with Trusted Computing (TC).

Grid computing has emerged as new field in the area of distributed computing. "The Grid" provides flexible, secure and coordinated access to shared resources among dynamically changing virtual organizations [12]. Much like the power grid, it aims to make computational power at the level of supercomputing an ubiquitous commodity resource which is available at low costs. Different projects use grid computing to tackle complex problems like climate prediction[1] or the search for collisions in the SHA-1 hash function[2]. Not only large organizations provide resources, but also individual users may donate unused CPU cycles, bandwidth or memory.

Such a heterogenous environment is ideally suited for virtual machine environments [13] with middleware based on Java commercially available. Java is an object-oriented, type-safe software environment with built-in security mechanisms such as access control, signed executable code and support for cryptography. Together with the security mechanisms, the intermediate byte-code representation of Java programs provides a "natural" isolation to other applications. It is portable per-se and also allows easy deployment. Building on the original notion of a sandbox, code can be executed with a defined set of access rights to system resources with flexible *stack inspection* mechanisms. Just-In-Time compilation creates native code for computational intensive program hot spots at runtime, thus mitigating the performance disadvantages of bytecode interpretation.

[1] http://www.climateprediction.net/
[2] http://www.iaik.tugraz.at/research/krypto/collision/index.php

Grid Computing as a Use Case for Trust Services The general operation model we consider as possible use case in this paper is as follows. A grid user identifies a problem which requires a large computational effort to solve. With the help of standard conformant middleware packages and libraries, she creates a software package. It is then automatically deployed to the participants on the grid. Work data is partitioned and distributed via secure channels to the remote systems, which are composed of a diverse selection of architectures. Work is processed, the results returned and assembled.

Of course, security in such a distributed system, where application code and data is sent to remote systems that are under control of another administrative domain, is critical.

In a current Grid system, the user, respectively the middleware she chooses to use, is required to trust the participants. The following security risks, as identified in [15], arise from this.

I Participants may deliberately report back wrong answers.
II Participants may not enforce data integrity or not compute accurately.
III Data protection requirements of sensitive data cannot be enforced on remote systems.
IV Theft of intellectual property on the distributed code by the participants is possible.
V Theft of confidential data by the participants is possible.

1.1 Related Work

A general study on the applicability of TC for grid computing, including use cases, is given in [15]. In [22], integration of VMs with grid computing is categorized and discussed how the policy and sandboxing features of an assumed-trustworthy Java Virtual Machine (JVM) can be used. TC is only suggested as an approach to integrate legacy systems in future grids. In the so-called Trusted Grid Architecture [20] participants integrate grid functionality in a hardware-supported virtualization scheme. Native code grid jobs are run in compartments[3] separated from a legacy OS. A protocol is designed for the attestation of multilateral security. Likewise, in the Daonity [23] project, native virtualization and TC are applied to Grid middleware. Platform independence or the issues that arise when realizing trust and measurements within virtual machines are not considered in these works.

Apart from work in grid computing, several approaches in the area of attestation are relevant to this work. The concept of *Property-Based Attestation* (PBA) [5] 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. As the certification is done externally, that approach is called *delegation*. Chen et al. [19] have proposed a protocol for property-based attestation, also following the delegation approach. They identify two additional categories to determine properties which are *code control* and *code analysis*. Code control infers regulations on the target machine, e.g. using SELinux as a reference monitor [30]. The attestation includes SELinux and its security policy. An example for code analysis is *Semantic Remote Attestation* (SRA) [6]. SRA utilizes language-based techniques to attest high level properties of an application running in a JVM. The approach is hybrid as it uses binary attestation to attest to the *Trusted Computing Base* (TCB) below the JVM. However, detailed knowledge of the application and eventual protocols is needed to extract high-level properties. A generalized approach based on a language-based VM has not been proposed yet.

[3] Note that such hardware-emulating compartments are often referred to as Virtual Machine (VM). In this paper we use the term for intermediate code interpreters like the Java VM exclusively.

1.2 Contribution

With the contributions of this paper we present novel Trust Services for language-based Virtual machines. We show how the afore-mentioned risks in the context of Grid computing can be mitigated as an exemplary use-case. We present a JVM with integrated services for TC based on a Trusted Computing Platform (TCP). Those services allow to extend the chain-of-trust into a language-based VM environment and their transparent usage in a remote attestation scenario. To complete the integration of TC into Java, we present the outline of an API allowing applications to use functionality such as key-management and provide security for application data. We believe this separation of concerns, low-level security services and a high-level API for applications, provides the necessary flexibility to tackle security problems arising in complex architectures, such as found in grid-computing. As our approach is based on the Java programming language, we maintain platform independence.

The JVM services presented in this work include support for property-based attestation. We see sandboxing as a tool to achieve security guarantees for grid computing and propose to extract properties from the security policy of the JVM. This allows us to provide a flexible basis for a framework for generalized attestation and sealing. We further outline a straightforward way to integrate those properties into existing proposals of protocols for property-based attestation.

1.3 Outline of the Paper

The remainder of this paper is organized as follows. Approaches to provide a trustworthy execution environment for the Java bytecode interpreter are discussed in Section 2. In Section 3 we present the extension of the so created *chain-of-trust* into the JVM. Section 4 explains how hardware supported TC functionalities can be provided to platform independent applications. The paper concludes in Section 5.

2 Trustworthy execution of Virtual Machines

Security enforced by software can be manipulated by software based attacks. To overcome this dilemma, the Trusted Computing Group (TCG) [10] has defined the TPM, a separate chip affixed to the mainboard of the computing platform. Similar to a smartcard, the device provides cryptographic primitives such as a random number generator, asymmetric key functionality and hash functions. A non-volatile and tamper-proof storage area keeps data over reboots. Access to the TPM is controlled through authorization of the commands and input data using a secret known to the *owner* of the TPM. The chip itself is a passive device providing services to software running on the main CPU, but does not actively take control or measurements of the host system.

Instead, the TPM allows to record measurements of the platform state in distinct Platform Configuration Registers (PCR). It receives measurements x from system software and hashes the input to the PCR with index i and content PCR_i^t using the *extend* operation

$$\mathrm{PCR}_i^{t+1} = \text{SHA-1}(\mathrm{PCR}_i^t, x).$$

At a later time the TPM can provide a compelling proof by *attesting* the received measurements in the form of *quoting* a set of PCR contents, cryptographically signed with a private Attestation Identity Key (AIK) key bound to the TPM. This integrity report allows a remote party to form a qualified trust decision of the platforms state. Data may also be bound to PCRs, in a process called *sealing*, in which decryption is only possible in a single valid system state.

At current level of technology, it is impossible to provide an full security assessment for open architectures like todays personal computers. Thus, the TC concept does not claim to enforce

perfect security under all conditions and tasks. The TCG defines a trustworthy system as *a system that behaves in the expected manner for the intended purpose.*

A system may provide a proof report of its state by applying the *transitive trust* model. Starting from booting the hardware with the BIOS acting as the root of trust each successive software block is measured into the TPM before it is executed, leading to a *chain-of-trust.* There are variations of this concept. The basic TCG model envisions a *static* chain-of-trust from hardware reboot onwards. Newer developments in CPUs and chipsets provide the option of a *dynamic* switch to a trusted system state: A special CPU instruction switches the system into a defined security state and then runs a measured piece of software which assumes full system control. Close hard-wired cooperation of CPU, chipset and TPM guarantees that the result is accurate. Using a hypervisor with support for hardware enforced virtualization, for instance Xen[4], allows execution of trusted and untrusted code in dedicated isolated compartments.

An intuitive assessment of a systems security is its size: The larger a system is the more likely it contains security relevant problems. Additionally, measuring of code takes time, especially when using the TPM itself with its low-performance communication link and hash implementation. Thus it is desireable to minimize the Trusted Computing Base, the critical parts of a system where every small bug can compromise global system security. Using a JVM as trust basis, the question of possible size reduction of the OS layer below arises. This issue has been addressed by Anderson et al. [31] who created a library OS for the Xen hypervisor.

With a choice of the mechanisms above, depending on the actual architecture, the JVM can be executed in a trustworthy way, providing a platform-independent TCB.

3 Trusted Computing Services for the JVM

We now address issues related to the JVM itself, starting with the basic security services provided by a language-based VM environment.

Java provides security mechanisms based on a security policy that enforces access control through stack inspection [24,21]. That is, the decision whether a method is allowed to access a resource is based on the entire call stack. The algorithm searches each stack frame starting from top to bottom (hence starting from the most recently called method) and tests whether the method has the appropriate *permissions* to access the resource. Each stack frame is mapped to a *protection domain* through the class the method belongs to. Each protection domain has assigned a set of permissions[5] which is checked through a *security manager* interface.

The security manager is held by the `System` class. A program fragment that needs access to a security relevant resource has to query the *System* class, check whether the security manager is actually instantiated (i.e. whether the return value is not `null`), and then query the security manager interface for access rights on the resource. The access control model is configured in so-called *policy files* at the granularity of code-bases (i.e. locations on the class path).

3.1 Code Identity

Code identity, based on binary measurement, is the central metric for remote authentication as defined by the TCG. The identity of the code is obtained through applying the *extend* command to binary executables. For a Java environment this would be class files, and as a special case, JAR

[4] `http://www.cl.cam.ac.uk/research/srg/netos/xen/`

[5] The full set of permissions available in the OpenJDK is documented in http://java.sun.com/javase/6/docs/technotes/guides/security/permissions.html

files. A JAR file is the typical unit of deployment for an application or a library and contains a collection of classes and associated meta-data.

At run-time, the JVM searches for classes on the so-called *class path* and dynamically loads them into the environment. As Java aims at a networked environment, class path entries may include remote locations as well. Hence, the code that is running on a JVM is generally not known on beforehand. We implemented a measurement architecture for the JVM which performs measurement of class files (resp. JAR files) and places the hashes in a PCR. To this end, we integrated measurement hooks into the class loading process. This allows us to extend the chain-of-trust into the JVM and gives the possibility to integrate further TCG concepts into the Java environment. In the context of Grid computing, this allows the user to verify that the code executed by the participant is actually identical to the distributed package, thus dealing with security risks I and II.

The JVM allows application designers to swap out certain operations to native code, i.e. to use native libraries from Java programs. This feature is intended for low-level access to the hardware and for computationally intensive algorithms. An example is the `System.arraycopy` operation from the runtime library which performs better than any Java equivalent in terms of performance. As such a library directly accesses VM memory, it is a potential security problem for the measurement architecture. We deal with this problem through measuring the code of the native library.

3.2 Trust Properties from the VM Configuration

The binary attestation approach foresees that configuration files of applications are measured and, to allow the verifier determine the security of the configuration, are included in the Stored Measurement Log (SML). As discussed before, the configuration of the access control mechanism is done in a *java policy* file. On a remote verifier side, this policy file has to be evaluated and checked whether it conforms to a given security policy, i.e. whether it matches the verifier's needs. As the policy files are not of a precisely defined format, identical configurations may hash to different PCR values. Hence, there needs to be a non-ambiguous mean to determine the *properties* of a specific configuration.

A work by Sheehy et al. [17] defines delegation as a requirement to ensure the trustworthiness of a generalized attestation mechanism. We address this issue in the following way: Determining the security of the configuration of the JVM can be delegated to a TTP through binary attestation. However, to obtain properties of that configuration, the TTP needs to translate the information as well. The proposed approach can also be used at a TTP to translate the configuration into a set of properties.

The Java security model permissions related to the specific scenario of Grid computing demand evaluation. To this end, we group certain permissions of the security policy to obtain a set of *trust properties*. The list of properties we propose is shown in Table 1. Note that this approach is an "all-or-nothing" approach. For instance, the property *file access* tells an attestor whether any code is able to access the file system. Further restrictions apply for the other properties. We have introduced this abstraction of the configuration to reduce the complexity of the access control model of Java as a first step towards determining properties.

We map permissions related to the general run-time security features to a single trust property *runtime*. An attestor has the possibility to determine whether the security manager is activated, and whether an application has the ability to change its implementation, the installed security policy, and class loaders. Furthermore, the settings of security permissions are reflected.

The full set of properties allows us to deal with several issues:

Table 1. Trust properties for a JVM security policy for use in grid computing.

Trust Property	Semantics	Java Permissions
Runtime	Status of run-time security features	`createSecurityManager`, `setSecurityManager`, `modifyThread`, `modifyThreadGroup`, `stopThread`, `createClassLoader`, `getClassLoader`, `setContextClassLoader`, `enableContextClassLoaderOverride`, `suppressAccessChecks`, any security permission
Native code	Ability to use native libraries	`loadLibrary`
File access	Ability to access the file system	any file permission
Network access	Ability to access the network	any network/socket permission

- Sensitive Data: If a specific JVM instance has the property of *file access* set to `false`, data cannot be written to disk, and hence, not be gathered by other, malicious entities. We, of course, assume that the network channel itself is confidential which is a property as well.
- Native Code: in special cases hardware optimization of the computing algorithm might be necessary. In addition to the bytecode, binaries can be distributed for invocation via JNI. As we are able to measure these files as well before executing them, this does not reduce the overall security. Grid applications using native code need the respective property to be `true`.

The proposed properties of the security policy are examples which match the needs of grid computing and allow us to enforce a behavioral policy to handle risk III. In [18], a property P is defined as a bit-string s of length l. As we chose to map permissions to properties that can only be `true` or `false`, l of such properties can be integrated into a single bit-string of respective length. This concept can be extended by using several bits for a single property to allow for more flexibility in defining s. Thus, the complex Java policy file can be reduced to s which can easily be extended to a PCR and thus integrated in attestation protocols, such as a binary attestation protocol or a property-based attestation protocol as proposed in [19].

3.3 Trusted Class Sealing

The distribution of software packages, as on the grid, might endanger intellectual property of the user, for instance, when the computational algorithm itself is of value. Current middleware [14] relies on bytecode obfuscation which makes recovery of the original source code only more difficult but not impossible. An other naïve approach, to encrypt class files and decrypt in a special class loader is easy to attack [26] if the current system integrity is not considered. Thus, to handle security risk IV we propose the concept of *trusted class sealing*.

This should not be confused with the following legacy concepts: Firstly, package sealing, which is a mechanism to control object inheritance [28]. Secondly, object sealing [29], allows one to encrypt all serializable objects, but without hardware protection of the keys and not transparently to the application. Instead, the user seals the sensitive classes to a trusted state of the target environement. Unsealing a class can be done through decoding the byte-stream in the class loading process. For performance reasons, only individual, security sensitive, classes should be sealed.

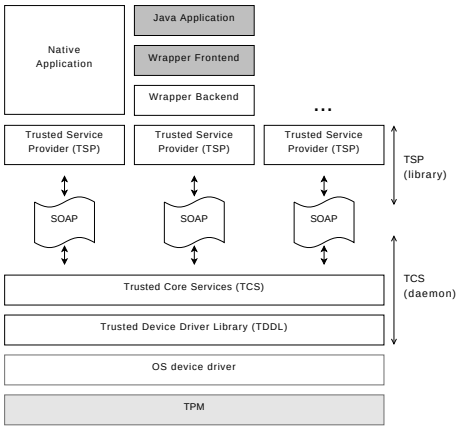

Fig. 1. Overview of jTSS Wrapper layered architecture, allowing both integration in Java and concurrent access. Legend: Hardware is light gray, native code is white and Java code is dark grey.

4 Application Interfaces to TC Mechanisms

The trustworthiness of a system does not end at the mechanics internal to the JVM, but extends into the application layer as well, especially to the maintenance and control middleware of a Grid. To receive sealed data and decrypt it, a participant requires both, high-level networking capabilities as provided by Java and access to TC services. The same holds true for remote attestation, where a networked protocol needs to maintain and transmit both the high-level SML and the PCR states from the TPM. It is a small step to allow also general purpose Java applications the access to TC services based on the TPM hardware device by means of defining an open Application Programming Interface (API).

There are two distinct approaches to integrate such an interface into Java. Firstly by *wrapping* an existing service of the native OS. A second approach is to create a complete *pure Java* TPM access software.

As the TPM is a device with limited resources a singleton software component has to manage it. Besides the VM, as detailed in Section 2, other software such as the OS require TPM services as well, possible already at boot time. Combining the need for an interface that handles both, resource management and concurrent access, makes it clear that this service cannot, in general, be implemented on top of the stack of running software alone, i.e. within a virtual machine.

Yet in case of a fully virtualized environment [25], where an exclusive environment for Java is provided in a compartment of its own and no concurrent accesses occur, the pure Java approach, as detailed in [16] or used by [2], will allow one to reduce the overall size and complexity. Such implementations may also be used if the resources of the TPM alone are virtualized as, for instance, with the TPM Base Services (TBS) [3] in Windows Vista. Still, for grid computing wrapping has the major advantage that it allows deployment on legacy operating systems as found today, which do not provide TPM virtualization. In the remainder of this section, we present the wrapper approach. An overview of the architecture is given in Figure 1.

The TCG Software Stack The TCG not solely specifies TPM hardware, but also defines an accompanying layered software infrastructure called the TCG Software Stack (TSS) [8]. An

interface in the C programming language allows applications to access TC functionality in a standardized way.

Generic TPM 1.2 drivers are integrated in recent OS releases, like Linux or Windows Vista. The lowest layer of the TSS, the *Trusted Device Driver Library (TDDL)*, abstracts the drivers to a platform independent interface that takes commands and returns responses as bytestreams. System tools provide basic functionality like resetting or taking ownership.

Since the hardware resources of the TPM are limited, the loading, eviction and swapping of keys and authorization sessions need to be managed. This is implemented in the *Trusted Core Services (TCS)*, which run as a singleton system service for a TPM. Additional functionalities are persistent storage of keys, TPM command generation and communication mechanisms. The TCS event manager handles the SML where PCR extend operations are tracked. To upper layers, a platform-independent Simple Object Access Protocol (SOAP) interface is provided. It is designed to handle multiple requests by ensuring proper synchronization.

System applications can access Trusted Computing functionality by using the *Trusted Service Provider (TSP)* interface. It provides a flat C-style interface. A `Context` object serves as entry point to all other functionality such as TPM specific commands, policy and key handling, data hashing, encryption and PCR composition. In addition, command authorization and validation is provided. Each application has their own instance of the TSP library, which is communicating via SOAP to the underlying TCS.

Java Bindings for the TSS The jTSS Wrapper software package integrates the TSP service into Java. The Java Native Interface (JNI) is used to call the functions of the TSP. A thin *C Backend* integrates the TSP system library, in our case that of the open source TrouSerS implementation [4]. From it, the JNI link is in large parts autogenerated using the SWIG[6] tool, which creates a set of Java methods out of the C functions and structures. Building on this, the *Java Frontend* abstracts several aspects of the underlying library, such as memory management, the conversion of error codes to exceptions and data types. On top of it all an API is provided to developers. Being a slight abstraction of the original C interface it permits to stay close to the original logic and provides the complete feature set of the underlying TSS implementation.
With features like data sealing and unsealing available to the distributed software, risks like V can be handled and protocols for risks I-III implemented.

5 Conclusions and Outlook

In this paper we outline approaches to increase the security of JVM based computing using Trusted Computing concepts. A chain of trust can be built from boot-up to the execution of the JVM. We propose how property- based attestation can be realized scenarios by deriving abstract security properties from the security policy configuration of the JVM. We show how to create a foundation for remote attestation of grid participants to the user.

With the concept of TPM based sealing of classes we propose a solution of how TC can be applied to protect intellectual property of distributed code. To actually allow middleware as well as general purpose applications access to TPM features, such as data sealing, a software interface is presented. It integrates in todays environments by wrapping existing TC services of the host. This allows a Grid user to establish trust in the participants of a grid.

Our proof-of-concept implementation is based on Sun's OpenJDK and openly avaiable at [7]. It currently encompasses an implementation of binary measurement as outlined in Section 3.1. Our experiments with the measurement architecture show that measurement of single class files

[6] `http://www.swig.org/`

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.

Currently, we do not consider the Java notion of signed code, a feature orthogonal to TC. Furthermore, to reduce the complexity, we do not differentiate trustworthiness of code-bases, i.e. different locations of the classpath. We leave these issues open for future work.

The original TSS design strives to provide access to the bare functions of a TPM and introduces only a few high-level abstractions. It is specified in and following the conventions of C. As our Java API focusses on providing feature completeness, its behavior is nearly identical. Repeated passing of numerous parameters and constants in hardly varying call sequences, instead of objects with intuitive behavior are witnessed. It would be more beneficial, if an easier to use, well-abstracted and object-oriented API were available for Java. We are currently designing and standardizing such a future high-level API [27].

Acknowledgements The authors thank anonymous reviewers for giving comments on an earlier version of this paper.

The efforts at IAIK to integrate TC technology into the Java programming language are part of the OpenTC project funded by the EU as part of FP-6, contract no. 027635. The project 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*, pp. 223–238, USENIX Association, 2004.
2. 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.
3. Microsoft Developer Network. TPM Base Services. `http://msdn2.microsoft.com/en-us/library/aa446796.aspx`, 2007.
4. TrouSerS - An Open-Source TCG Software Stack Implementation. `http://trousers.sourceforge.net/`, 2007.
5. 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)*, pp. 67–77, ACM, 2004.
6. 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*, pp. 29–41, USENIX Association, 2004.
7. M. Pirker, T. Winkler, R. Toegl and T. Vejda. Trusted Computing for the Java™Platform, `http://trustedjava.sourceforge.net/`, 2007.
8. Trusted Computing Group. TCG Software Stack Specification, Version 1.2 Errata A. `https://www.trustedcomputinggroup.org/specs/TSS/`, 2007.
9. Trusted Computing Group. TCG Infrastructure Specifications. `https://www.trustedcomputinggroup.org/specs/IWG`, 2007.
10. Trusted Computing Group. `https://www.trustedcomputinggroup.org`, 2007.
11. Trusted Computing Group. TCG Specification Architecture Overview, Revision 1.4, `https://www.trustedcomputinggroup.org/groups/TCG_1_4_Architecture_Overview.pdf`, 2007.
12. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int. J. High Perform. Comput. Appl.* 15(3):200–222, Aug. 2001.

13. V. Getov, G. von Laszewski, M. Philippsen and I. Foster. Multiparadigm communications in Java for grid computing. *Communincations of the ACM* 44(10):118–125, Oct. 2001.
14. Parabon Computation, Inc. Frontier: The Premier Internet Computing Platform Whitepaper. `http://www.parabon.com/users/internetComputingWhitePaper.pdf`, 2004.
15. W. Mao, H. Jin and A. Martin. Innovations for Grid Security from Trusted Computing, `http://forge.gridforum.org/sf/go/doc8087`, 2005.
16. K. Dietrich, M. Pirker, T. Vejda, R. Toegl, T. Winkler and P. Lipp. A Practical Approach for Establishing Trust Relationships between Remote Platforms using Trusted Computing. In *Proceedings of the 2007 Symposium on Trustworthy Global Computing*, in print, 2007.
17. J. Sheehy, G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, L. Monk, J. Ramsdell, and B. Sniffen. Attestation: Evidence and Trust. Technical report 07 0186, MITRE Corporation, 2007.
18. U. Kühn, M. Selhorst, and C. Stüble. Realizing Property-Based Attestation and Sealing with Commonly Available Hard- and Software. In *Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing*, pp. 50–57, ACM, 2007.
19. L. Chen, R. Landfermann, H. Löhr, M. Rohe, and A.-R. Sadeghi. A Protocol for Property-Based Attestation. In *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, pp. 7–16, ACM, 2006.
20. H. Loehr, H. Ramasamy, A.-R. Sadeghi, S. Schulz, M. Schunter, C. Stueble. Enhancing Grid Security Using Trusted Virtualization. In *Proceedings of the 4^th Conference on Autonomic and Trusted Computing (ATC-07)*, pp. 372–384, Springer-Verlag, 2007.
21. D. Wallach and E. Felten. Understanding Java Stack Inspection. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pp. 52–63, IEEE, 1998.
22. M. Smith, T. Friese, M. Engel, B. Freisleben. Countering security threats in service-oriented on-demand grid computing using sandboxing and trusted computing techniques. *J. Parallel Distrib. Comput.* 66(9):1189–1204, Sept. 2006.
23. W. Mao, F. Yan and C. Chen. Daonity: grid security with behaviour conformity from trusted computing. In *Proceedings of the First ACM Workshop on Scalable Trusted Computing (STC'06)*, pp. 43–46, ACM, 2006.
24. L. Gong, M. Mueller, H. Prafullchandra, and R. Schemers. Going beyond the sandbox: an overview of the new security architecture in the javaTM development Kit 1.2. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pp. 103–112, USENIX Association, 1997.
25. S. Berger, R. Cáceres, K. Goldman, R. Perez, R. Sailer, L. van Doorn. vTPM: Virtualizing the Trusted Platform Module. IBM Research Report, RC23879 (W0602-126), 2006.
26. Roubtsov, V. Cracking Java byte-code encryption, JavaWorld. `http://www.javaworld.com/javaqa/2003-05/01-qa-0509-jcrypt_p.html`, 2003.
27. R. Toegl et al. Trusted Computing API for Java, Java Specification Request 321, Java Community Process, `http://www.jcp.org/en/jsr/detail?id=321`, 2008.
28. M. Biberstein, J. Gil and S. Porat. Sealing, Encapsulation, and Mutability. In *Proceedings of the 15th European Conference on Object-Oriented Programming*, pp. 28–52, Springer-Verlag, 2001.
29. L. Gong, and R. Schemers. Signing, Sealing, and Guarding Java Objects. In *Mobile Agents and Security*, pp. 206–216, Springer-Verlag, 1998.
30. T. Jaeger, R. Sailer, and U. Shankar. PRIMA: policy-reduced integrity measurement architecture. In *Proceedings of the eleventh ACM symposium on Access control models and technologies (SACMAT'06)*, pp. 19–28, ACM, 2006.
31. M. J. Anderson, M. Moffie, and C. I. Dalton. Towards Trustworthy Virtualisation Environments: Xen Library OS Security Service Infrastructure. HP Research Report, HPL-2007-69, 2007.