Enhancing Grid Security Using Trusted Virtualization[1]

Hans Löhr[‡] HariGovind V. Ramasamy[†] Ahmad-Reza Sadeghi[‡]
Stefan Schulz* Matthias Schunter[†] and Christian Stüble[‡]

[†]IBM Zurich Research Laboratory
Rüschlikon, Switzerland
{hvr,mts}@zurich.ibm.com

[‡]Horst-Görtz-Institute for IT-Security
Ruhr-University Bochum, Germany
{loehr,sadeghi,stueble}
@crypto.rub.de

*Max-Planck Institut für Eisenforschung, Germany schulz.stefan@gmail.com

Abstract. Grid applications increasingly have sophisticated functional and security requirements. Current techniques mostly protect the grid resource provider from attacks by the grid user, while leaving the user comparatively dependent on the well-behavior of the provider. We present the key components for a trustworthy grid architecture and address this trust asymmetry by using a combination of trusted computing and virtualization technologies. We propose a scalable offline attestation protocol, which allows the selection of trustworthy partners in the grid with low overhead. By providing multilateral security, i.e., security for both the grid user and the grid provider, our protocol increases the confidence that can be placed on the correctness of a grid computation and on the protection of user-provided assets.

1 Introduction

Grid computing has been very successful in enabling massive computing efforts, but has hitherto been dominated by 'big science.' These projects are usually in the academic domain (such as SETI@HOME or distributed.net) and, although important, they usually have less stringent security requirements than commercial IT systems. Currently, security is built into grid toolkits (e.g. the Globus toolkit [1]) used at the *provider* sites (parties that offer resources for use in the grid). Secure channels, authentication, unsupervised login, delegation, and resource usage [2] are all handled by the toolkit. These mechanisms usually do not protect the grid *user* (the person or entity wishing to utilize resources).

The user is forced to trust[2] the provider, often without the possibility of verifying whether that trust is justified. However, in much of the current literature on grid security (e.g., [4]), the user is not regarded as trustworthy. This trust asymmetry could potentially lead to a situation in which the grid provider causes large damage to the user with little risk of detection or penalty. An attacker might publish confidential data or sabotage the entire computation by providing false results. These problems are most evident in computational grids, especially in mobile code [5] scenarios. Other grids, such as storage or sensor grids, may also suffer from the negative consequences of this trust asymmetry. Because of this problem, companies are reluctant to utilize available grid resources for critical tasks.

Given this state of affairs, Mao et al. [6] have advocated the use of the emerging *Trusted Computing* (TC) technology for the grid. In a similar vein, Smith et al. [7] more closely examine scenarios

[1] A preliminary version of this work was presented (without publication) at the *2nd Workshop on Advances in Trusted Computing 2006* and at the *1st Benelux Workshop on Information and System Security 2006*.

[2] In this paper, we consider "trust" to be the opposite of enforcement. Thus, a trusted component is a component whose well-behavior cannot be enforced by another component and, therefore, has the capability to violate a security policy. This view of trust contrasts with the notion put forward in other grid-related works, such as [3], which view trust as a positive, reputation-based property.

that could benefit from TC techniques. TC can be used to enforce *multilateral security*, i.e., the security objectives of all parties involved are taken into account.

A trustworthy grid environment that enforces multilateral security would offer a number of benefits. Even sensitive computations could be performed on untrusted hosts. Most personal computers used today possess computing abilities in excess of what is required for casual or office use. These resources could be leveraged to run grid jobs in parallel to the users' normal workflow and provide the computational power necessary for next-generation modeling and simulation jobs, without costly investments into new infrastructure. Enterprises could utilize the already-present office machines more fully, resulting in an earlier return on their investment.

A large percentage of the platforms in large-scale grids are built using general-purpose hardware and software. However, it is easy and cheap for existing platforms to incorporate a *Trusted Platform Module* (TPM), based on specifications of the *Trusted Computing Group* (TCG). The module provides a trusted component, usually in the form of a dedicated hardware chip. The chip is already incorporated into many newly-shipped general-purpose computers. The TPM chip is tamper-evident (and ideally, tamper-resistant) hardware that provides cryptographic primitives, measurement facilities, and a globally unique identity. For verification purposes, a remote party can query the TPM's measurement of the *Trusted Computing Base* (TCB) by means of *attestation*. This mechanism, proposed by the TCG, enables (remote) verification of the status of a platform's TCB.

One approach to securing computing systems that process potentially malicious code (such as in many number-crunching grid applications) is to provide a virtualized environment. This technique is widely used for providing "V-Servers," i.e., servers running several virtual machines that may be rented to one or several users. Although users have full control over the virtual environment, they cannot cause damage outside that environment, except possibly through attempts at resource monopolization, for example, by "fork bombing." Although virtualization offers abstraction from physical hardware and some control over process interaction, there still are problems to be solved. For example, in the x86 architecture, direct memory access (DMA) devices can access arbitrary physical memory locations. However, hardware innovations such as Intel's Trusted Execution Technology [8] (formerly known as LaGrande) and AMD's Virtualization Technology [9] (formerly code-named Pacifica) aim to address these problems and could eventually lead to secure isolation among virtual machines. Virtualization technology can be leveraged for building a trustworthy grid environment, especially because several works, such as [10], have already begun to consider architectures that feature policy enforcement in the virtualization framework.

Our Contribution. To address the trust asymmetry in grid computing explained above, we propose a realistic security architecture that uses TC functionality and enforces multilateral security in a grid scenario. Leveraging a combination of the isolation (between virtual machines) provided by virtualization and a trusted base system, our design is able to protect confidentiality and integrity in a multilateral fashion. We feel our compartmented security design offers a stronger level of protection than many current techniques can provide.

Using our security architecture, we propose a grid job submission protocol that is based on offline attestation. The protocol allows a user to verify that a previously selected provider is in a trusted state prior to accessing a submitted grid job, with little overhead and improved resistance to attack. Our protocol also guarantees *transitive* trust relations if the provider in turn performs further delegations to other providers.

2 Preliminaries

2.1 System Model and Notation

We consider the following abstract model of the grid. A grid user `U` can attempt to access any grid provider `P`. Each participant in the grid is considered to be a partner-and-adversary that potentially intends to harm other participants but also provides services. A participant can be depended upon to execute a given task correctly only if it can prove its inability to cause damage (break a partner's security policy).

A machine $\mathtt{m}$ is a single physical host. It can host one or more logical participants of either role. We consider delegation to be modeled as one participant being both a provider and a user. Every participant has its own, distinct policy. Each component of $\mathtt{m}$ is an independent actor offering some interface(s) to other components, and usually utilizing interfaces offered by other components. The set of providers and users need not be static, but can grow and shrink dynamically as new resources are being added to the grid virtual organization (VO), and some participants leave the VO. However, joining and leaving are not the focus of this paper.

For our purposes, a job image is a tuple $\mathtt{J} = (\mathtt{data}, \mathcal{C}, \mathtt{SP}_\mathtt{U})$, where $\mathtt{data}$ may be an invocation to some predefined interface or carry executable code. For security purposes, both input data and executable code have the same requirements and can be protected using the same techniques. Therefore, we do not distinguish between "code" and "data," and refer to both as $\mathtt{data}$. $\mathcal{C}$ represents the credentials of the user $\mathtt{U}$, which may be needed to gain access to the provider $\mathtt{P}$. The user also passes a policy $\mathtt{SP}_\mathtt{U}$ as part of its invocation, which specifies constraints to be upheld for that particular job. The job, once scheduled, can communicate directly with $\mathtt{U}$ (subject to the policy $\mathtt{SP}_\mathtt{U}$).

A machine $\mathtt{m}$ always has exactly one state σ describing the status of the TCB rather than a particular VM. This state comprises all code running as part of the TCB. TCB components are critical to the correct functioning of the system and need to be trusted. Adding, removing, or modifying such a component changes σ. However, σ will not change because of "user actions," such as installing application software, browsing the web, or executing a grid job. Furthermore, the system will not allow any party (not even system administrators) to alter the TCB without changing σ. σ' is the reported state of the platform, possibly different from σ. We assume that σ and σ' can be encoded as a configuration (or metrics) $\mathtt{conf}$, a short representation of the state (e.g., a hash value) as determined by a *measurement* facility (e.g., the TPM) of the machine. A specific aspect of the user's security policy $\mathtt{SP}_\mathtt{U}$ is the $good$ set, which contains the $\mathtt{conf}$ values of all states σ considered to be trustworthy by that policy.

$\mathtt{K}$ denotes an asymmetric cryptographic key, with private part $s_\mathtt{K}$ and public part $p_\mathtt{K}$. $\mathbf{enc}_{p_\mathtt{K}}(X)$ denotes a piece of data X encrypted with a public key $p_\mathtt{K}$. $\mathbf{sign}_{s_\mathtt{K}}(X)$ denotes a data item X that has been digitally signed by a private key $s_\mathtt{K}$.

2.2 Usage Scenario

We consider the following scenario: When a node joins the grid, it generates and publishes an attestation token τ, which can be used by potential partners to obtain assurance about the node's trustworthiness. Grid users retrieve attestation tokens from different grid nodes and select a token indicating a configuration they are willing to trust. The selection decision is made offline, and incurs negligible overhead on the part of the user. Once an acceptable provider has been found, users can submit jobs that can only be read by the selected node in the configuration they consider as trustworthy. If the node has changed to another configuration, communication will fail.

The main advantage of this approach is that the creation of the attestation tokens is decoupled from the process of job submission, while still providing freshness. In addition, these tokens are transferable and their correct creation can be verified without interacting with their creators.

2.3 Requirements

In this paper, we focus on security requirements, namely *integrity* and *confidentiality*. Providing integrity means protection against unauthorized modifications. For instance, user $\mathtt{U}$ should not be able to alter aspects of provider $\mathtt{P}$ to elevate its privilege level. Similarly, $\mathtt{P}$ should be prevented from

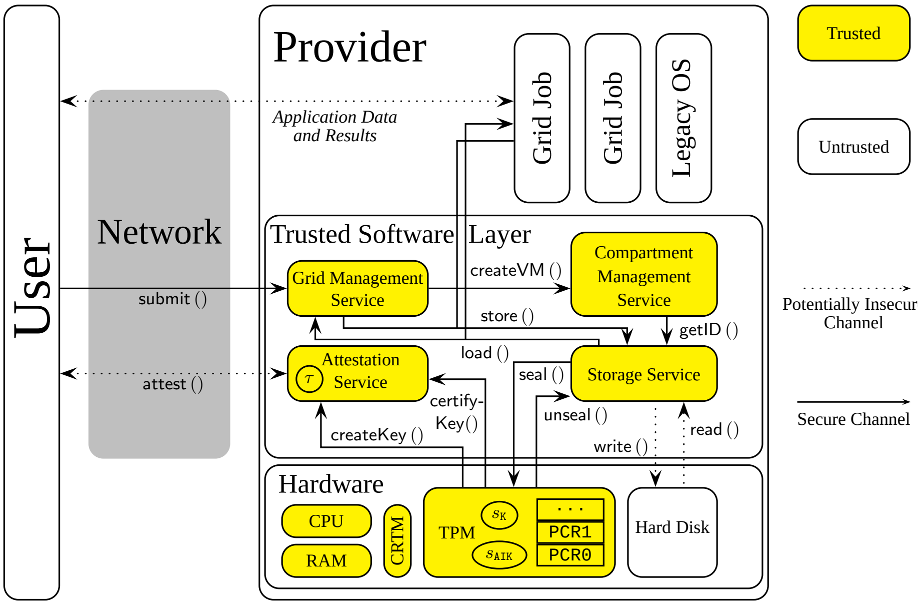


Fig. 1. Components of the Trusted Grid Architecture

modifying U's job. Both the user and provider may require confidentiality, i.e., they may require their sensitive data be guarded against unauthorized disclosure. U may utilize confidential data as part of J, and demand that this data not be disclosed to any party other than J's execution environment. Similarly, P may want to ensure that a malicious grid job cannot collect secrets stored on P's platform (such as signature keys) and forward them to U.

3 A Trusted Grid Architecture

Figure 1 shows the abstract building blocks of our Trusted Grid Architecture (TGA). The hardware platform provides a TPM and untrusted storage. The Trusted Software Layer (TSL) consists of the attestation, grid management, compartment management, and storage management components. The TSL provides both security functionalities and virtualization of the hardware. The TCB consists of the TSL and the trusted hardware components. Security policies have to be enforced by the TCB, but a detailed treatment of policy enforcement is outside the scope of this paper. Other works, such as [10] and [11], have examined some necessary properties of policy engines. Proper design of a *minimum* set of trusted services can help to achieve a TCB with the highest possible resistance to attacks. Additional guarantees about runtime behavior and state (e.g., [12]) may be provided by a dedicated service or as an extension to our attestation service.

We now provide an overview of the TGA components; more details can be found in [13].

Hardware: The core hardware component is a TPM as specified by the TCG, providing cryptographic functions such as encryption and signing. Each TPM possesses a number of platform configuration registers (PCRs), at least 16 as of version 1.2 of the specification [14]. During system boot, the main software components (BIOS, bootloader, OS kernel, etc.) are *measured*. The measurement procedure involves computing a configuration conf, i.e., the cryptographic hash of the software components, and securely storing the hash in the TPM. For the TGA, we use four TPM operations: secure key generation, measurement, certification, and sealing. The TPM features a hardware random-number generator and implements generation of RSA key pairs $\mathtt{K} = (p_{\mathtt{K}}, s_{\mathtt{K}})$. For these key pairs, usage limitations can be defined, in particular *sealing*, which marks the private key as not being migratable and usable only when a specified subset of the PCRs contain the same values as were present during key generation. It is possible to obtain a certificate stating which usage conditions apply to a key pair (as represented by its public key $p_{\mathtt{K}}$) from the TPM, signed by one of its *Attestation Identity Keys* (AIKs; generated by the TPM). The private key of an AIK cannot be extracted from the TPM, i.e., it is non-migratable, and it cannot be used to certify migratable keys. AIKs can be certified by a Certification Authority (CA), or they can be proved to be valid AIKs anonymously by means of Direct Anonymous Attestation (DAA) [15]. Such a certificate or proof is denoted as $\mathtt{cert}_{\mathrm{CA}}\,(p_{\mathtt{AIK}})$.

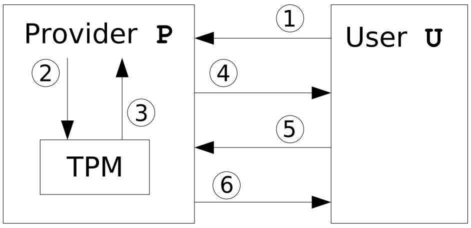
The TPM can report the platform configuration to other parties by signing the values of the PCRs with an AIK, which guarantees that the TPM generated the signed structure because an AIK cannot be used to sign arbitrary data. For our purposes, we use signed `KeyInfo` structures that are considered as certificates. A `KeyInfo` structure of a sealed key includes the selection of PCRs that were used for sealing, their values at the time of key generation, the values of the selected PCRs needed to use the sealed key (i.e., the `conf` of reported state σ'), and an indication whether a key is migratable. We use an AIK to sign such a structure with the certifyKey operation of the TPM and denote the resulting certificate by $\mathtt{cert}_{\mathtt{AIK}}\,(p_{\mathtt{K}})$. These restricted keys enable *data sealing*. Data sealed to a certain configuration of the system is encrypted with a public key whose corresponding private key is accessible only to a certain state and platform. If the data is successfully decrypted, this indicates that the state the key was sealed to is the actual state of that machine.

Attestation Service (AS): The AS provides metrics about the state σ to remote parties by means of an *attestation token* $\tau := (p_{\mathtt{AIK}}, p_{\mathtt{K}}, \mathtt{cert}_{\mathrm{CA}}\,(p_{\mathtt{AIK}}), \mathtt{cert}_{\mathtt{AIK}}\,(p_{\mathtt{K}}))$. From `conf` (contained in $\mathtt{cert}_{\mathtt{AIK}}\,(p_{\mathtt{K}})$), the user `U` is able to distinguish a trusted state σ' from an untrusted one because the values uniquely identify a set of programs that have been loaded since booting the platform, and possibly also the state of certain critical configuration files. The certificate $\mathtt{cert}_{\mathtt{AIK}}\,(p_{\mathtt{K}})$ identifies the key `K` as being sealed to `conf` and gives the assurance that the private key $s_{\mathtt{K}}$ can be used only in the reported state σ'. The user `U` can make its trust decision “offline” by examining the `conf` contained in τ. If `conf` is indicative of a trusted state σ', $s_{\mathtt{K}}$ will be accessible to the provider `P` only if `P` still is in the same configuration. As the token does not change over time, it can be distributed to other parties. If the state σ of `P` ever changed, τ would automatically become invalid, although an explicit revocation might still be beneficial. Further details of this attestation mechanism and its security will be discussed in Section 4.

Compartment Management Service (CMS): This component creates *virtual machines* (VMs; also called *compartments*), which run on top of the TCB, and keeps track of the identity of compartments by assigning a unique identifier (`ID`) to each of them. The VMs are isolated from each other and can only communicate over well-defined interfaces. The CMS only manages VMs locally and does not address migration or delegation in the grid.

Storage Service (SS): The storage component provides trustworthy and non-volatile storage based on an untrusted hard disk. In particular, data stored by one compartment in one configuration is retrievable only by that compartment in the same configuration – even if the machine has entered an untrusted state in the meantime. To achieve this property, all data is encrypted and MAC-authenticated by a sealed key.

Grid Management Service (GMS): The GMS handles the actual grid job submission. It is responsible for receiving jobs, checking their access, and instantiating them. It will use the CMS to create a private compartment for each job. The GMS does any special pre-processing that the job needs before it is ready for execution. Once such pre-processing has been done, a VM image has been created from `J`, which can then be booted by the CMS. Furthermore, the GMS takes the policy of the user and notifies an enforcement component (not shown in Figure 1) of the restrictions and



Common input: attestation token
$\tau = (p_{\mathtt{AIK}}, p_{\mathtt{K}}, \mathtt{cert}_{\mathrm{CA}}(p_{\mathtt{AIK}}), \mathtt{cert}_{\mathtt{AIK}}(p_{\mathtt{K}}))$
$\mathtt{U}$**'s input:** job $\mathtt{J}$ and the accept set $good_{\mathtt{U}}$
$\mathtt{P}$**'s input:** accept set $good_{\mathtt{P}}$
TPM**'s input:** $(s_{\mathtt{K}}, \sigma'_{s_{\mathtt{K}}})$, current state σ

1. $\mathtt{U}$ verifies $\mathtt{cert}_{\mathrm{CA}}(p_{\mathtt{AIK}})$, $\mathtt{cert}_{\mathtt{AIK}}(p_{\mathtt{K}})$, and $\mathtt{conf} \in good_{\mathtt{U}}$.
   Upon verification, $\mathtt{U}$ randomly chooses nonces N and N', and a session key κ.
   $\mathtt{U}$ sends $\mathbf{enc}_{p_{\mathtt{K}}}(\kappa)$ and $\mathbf{enc}_{\kappa}(\mathsf{N})$ to $\mathtt{P}$.
2. $\mathtt{P}$ forwards $\mathbf{enc}_{p_{\mathtt{K}}}(\kappa)$ to TPM.
3. TPM decrypts κ if $\sigma = \sigma'_{s_{\mathtt{K}}}$ and returns κ to $\mathtt{P}$.
4. $\mathtt{P}$ decrypts N and sends $\mathbf{enc}_{\kappa}(\mathsf{N}, good_{\mathtt{P}})$ to $\mathtt{U}$.
5. $\mathtt{U}$ verifies N and whether $good_{\mathtt{P}} \subseteq good_{\mathtt{U}}$; upon verification, $\mathtt{U}$ sends $\mathbf{enc}_{\kappa}(\mathsf{N}', \mathtt{J})$ to $\mathtt{P}$.
6. $\mathtt{P}$ decrypts N' and $\mathtt{J}$, and sends N' to $\mathtt{U}$.
   $\mathtt{U}$ verifies N'.

Fig. 2. Submission Protocol submit ()

rights declared therein. It also handles the freshness verification of the attestation token τ when a job is submitted (described in Section 4).

4 A Protocol for Scalable Offline Attestation

Attestation is the process of securely reporting the configuration of a party to a remote challenger. The most commonly discussed type of attestation requires a remote challenger to provide a random nonce N, which is then signed (together with a hash over a subset of the current PCR values) by the TPM using an $\mathtt{AIK}$. As freshness is achieved by means of a random nonce, each interaction necessitates a new attestation (and thus, a new TPM-generated signature). However, TPM signature generation is slow, and TPM commands generally cannot be parallelized. In addition, without appropriate countermeasures, this technique could potentially be vulnerable to a race between a successful attestation and a change of state prior to further interactions depending on the trusted state. If the state of the system changes after attestation has concluded, but before any further interactions take place, this change would not be noticed by the remote party. Also, without connecting attestation to a PKI identity, an attestation challenge could be relayed to a trusted platform by an attacker (by forwarding the trusted platform's reply to the verifier).

Scalable offline attestation is intended to enhance some aspects of current attestation systems. Having an *attestation token* that can be distributed freely within the VO as an informational item is advantageous, because this token states the current configuration of a provider $\mathtt{P}$, without requiring the prospective user to interact with that provider right away. The user can collect such tokens over time, and select the most appropriate configuration offline. As such a token cannot guarantee freshness, some verification has to occur when the user contacts the provider of his choice. We propose a *sealed key* approach, in which the provider's TPM allows usage of the private key only if the provider is in the same state as the key was stored in. The approach partitions the verification of $\mathtt{P}$'s state into two phases: token creation and freshness verification.

A provider $\mathtt{P}$ creates an attestation token together with its TPM. The attestation service instructs the TPM to create a non-migratable key sealed to a collection of PCRs. Then, the attestation service uses the TPM's certifyKey operation to create a certificate $\mathtt{cert}_{\mathtt{AIK}}(p_{\mathtt{K}})$ with an AIK. The attestation service then constructs the attestation token τ from the public key $p_{\mathtt{K}}$, the certificate of this key, $\mathtt{cert}_{\mathtt{AIK}}(p_{\mathtt{K}})$, the public part of the AIK, $p_{\mathtt{AIK}}$, and a certificate of the AIK, $\mathtt{cert}_{\mathrm{CA}}(p_{\mathtt{AIK}})$. The private key $s_{\mathtt{K}}$ is accessible only in the provider's state at the time of token generation, σ', because the certification is done using the TPM-internal AIK, which cannot be misused, even by the platform owner. The attestation service then publishes the token. Publication of the attestation token τ in effect becomes an advertisement stating that a certain state σ' will be maintained at P.

The protocol shown in Figure 2 includes the actual submission of the job and addresses freshness verification. If the conf contained in the token is considered good by the user U, then U generates a symmetric session key κ and encrypts the key using $p_{\mathtt{K}}$. The session key can be decrypted by the provider's TPM only if its state still matches the state at the time of τ's creation, i.e., P's reported state σ'. Verification of P's ability to access $s_{\mathtt{K}}$ is sufficient to ensure that P is actually in the state that was advertised by conf. The rationale for including the session key is twofold. First, asymmetric cryptography is by orders of magnitude slower than symmetric methods. Second, the key's inclusion reduces the necessary TPM operations from the signature generation (in traditional schemes) to a single asymmetric decryption.

The submission protocol further guarantees *transitive* trust. As the job gets delegated from one provider to other providers, it is assured that each party that is entrusted with the job's data will satisfy the original submitter's requirements. This is done by ensuring that each platform X that gains control of the user U's job J must satisfy the condition, $good_X \subseteq good_{\mathtt{U}}$.

Extensions. In contrast to protocols like DAA [15], our proposed protocol does not feature any privacy guarantees. As the platform has to reveal its actual configuration, it is in effect exposing potentially sensitive information to another party. Integrating privacy guarantees into our proposal could be an interesting aspect for future research. To address some privacy issues and other well-known limitations of binary attestation, property-based attestation and sealing schemes (e.g., [16]) could be integrated into our TGA.

5 Security Analysis

Security of Offline Attestation. The offline attestation mechanism proposed in Section 4 is secure against man-in-the-middle attacks. If a user U seals a job to a trustworthy attestation token τ, only the platform in possession of the private part of key K can unseal the job, and only if it is in the state indicated by τ. An adversary cannot decrypt the job, even if it is running on the platform with the TPM that holds the private key, if conf (corresponding to the platform's current state σ) does not match conf$'$ contained in τ (corresponding to the platform's reported state σ'). Conventional techniques need to include additional verification (such as tying an AIK to a PKI identity) to achieve the same assurance as ours.

Delegation with transitive trust ensures that every provider P that gets a job J can only access J if the provider is in a state σ that is trusted by the original submitter U, i.e., $\mathtt{conf} \in good_{\mathtt{U}}$ (where conf corresponds to σ). Transitive trust is achieved during delegation without communication with the submitter because the provider that wishes to transfer a job attests other providers offline prior to transmitting the job. The delegating provider $\mathtt{P}_1$ acts as user of the new provider $\mathtt{P}_2$ and verifies that $good_{\mathtt{P}2} \subseteq good_{\mathtt{P}1}$, which immediately implies that $good_{\mathtt{P}2} \subseteq good_{\mathtt{U}}$. Hence, the policy of the new provider $\mathtt{P}_2$ is also acceptable to the original user. Moreover, offline attestation is secure against replay attacks, under the assumption that state changes can only occur between protocol runs. Replaying of old, trustworthy attestation tokens does not help an adversary: the TPM will not allow decryption if the current PCR values do not match the values the key was sealed against.

Our protocol has the following drawbacks. Like conventional attestation, our protocol is vulnerable to TPM compromises. A compromised TPM can expose the secret key to an adversary, which enables the adversary to attest to arbitrary states. Revocation of AIKs is necessary to limit the potential damage such attacks may cause. As with conventional attestation, another risk of offline attestation is corruption of the running TCB. If an adversary can corrupt the TCB while the system is running, it could change the system's state σ without changing the PCRs. Thus, σ would deviate from σ', but the TPM would still allow the sealed key to be used.

Integrity Protection. Because we can establish a secure (confidential and integrity-protected) channel from user U to provider P using standard tools such as TLS, we need not consider in-transit modifications. Thus, for the purpose of this analysis, P receives an unaltered job J. We need to consider two kinds of integrity requirements for that image: before being instantiated and while executing. As results are reported directly, their integrity can again be achieved by established solutions. If job execution is delayed by the GMS, the job image and policy are stored in trusted storage. The key of the storage service is stored sealed, which guarantees that access to it is granted only to the same job in the same system state. In an untrusted state, no access is granted. Therefore, if a piece of data X in the storage service is altered, the signature of that data item cannot be updated, and the modification is detected the next time the data is retrieved from the storage service. While job J is executing, the isolation properties of our system guarantee that no untrusted application can gain access to the memory regions assigned to J, and hence, integrity is guaranteed. Circumventing such barriers would require breaching the TCB, which would contradict our assumption. As the TCB is based on a virtualization layer, even attack scenarios like "blue pill" [17] are ineffective, because such rootkits can only virtualize conventional systems that do not use virtualization techniques themselves. However, even if such a system were able to virtualize a virtualization layer, it would either need to compromise the TCB, or it would have to be loaded before the TGA (and thus, be measured in the boot process).

Confidentiality Protection. The two mechanisms employed for protecting the integrity of stored data and in-memory data also protect confidentiality. The CMS enforces isolation between the VMs and foils *in-memory eavesdropping*, i.e., one process accessing data inside the virtual memory of another process. Sealing prevents untrusted configurations from decrypting data stored in non-volatile storage. Violating confidentiality implies breaching the TCB for the in-memory scenario, as the TCB enforces virtualization and therefore, limits each application to its own VM, whereas decrypting stored data outside of a trusted state would necessitate breaking the encryption scheme used, which we likewise consider infeasible.

6 Discussion

Integration of Legacy Systems. To maintain interoperability with legacy systems, we aim to provide the means to continue using applications designed for existing grid toolkits (such as Globus [1]), without giving up the advantages our architecture offers. One possible way for such an integration would be to provide an executable image for each toolkit supported. Whenever an invocation for a service using that toolkit is received, it is instantiated, and the request forwarded to that instance. However, the grid toolkit must be part of the TCB. After all, a malicious provider might use a good base configuration, and put all its attack code into a modified toolkit image. The attestation token τ should contain measurements of all execution environments available as "default installations" on the platform. Thus, the benefits of our proposal become applicable without forcing the user to significantly change its use of the grid. Alternatively, a grid job may consist of a full, bootable VM. While this is a radically different approach from traditional grid methods, it does not imply further trusted code, which is desirable to keep the TCB small and of low complexity.

Implementation. We have started implementing the core components of the TGA architecture in the PERSEUS framework [18], which is based on a micro-kernel with paravirtualized Linux. The framework's design allows its porting to other systems (such as Xen), and features a strong separation of responsibilities even among the TCB (by running services as separate compartments), which significantly simplifies verification. Prototypes of the core TGA components have already been demonstrated in the context of the ongoing OpenTC [19] and European Multilaterally Secure Computing Base [20] projects.

Related Work. Several authors have suggested methods to increase the reliability of grid computation without TC technology. For instance, task replication or the introduction of quiz tasks [21] to detect misbehaving providers aimed at protecting the integrity of the results of grid computations. However, these techniques are wasteful in terms of resources and often not resistant to multiple colluding adversaries. Using virtualization to improve grid security has been proposed in numerous works (e.g., [22]).

Sailer et al. [10, 23] investigated the possible enforcement of MAC policies at the level of the virtualization layer. Sailer et al. [24] also proposed an integrity measurement architecture for Linux. Such an architecture could be useful for the measurement and reporting of VM states in our TGA. Similarly, although the proposed system of Jaeger et al. [25] focuses on improving the integrity checking of SELinux, its underlying principles could be used for verifying the correctness of the Trusted Software Layer of our TGA.

The Daonity (e.g., see [26]) project aims to strengthen the grid security infrastructure by integrating TC technology into the Globus toolkit. However, as Mao et al. [26] remark, the current version of Daonity does not take the operating system into account. For instance, an administrator could bypass the TC-based security mechanisms. To prevent such attacks, a system architecture with virtualization on top of a security kernel, as we propose in this paper, could be used.

Recently, Cooper et al. [27] proposed a security architecture for delegation on the grid based on TC and virtualization technologies. They describe a delegation service for enforcing local and global delegation policies. Offline attestation techniques, such as the one we propose, may be useful for their delegation service, whereas our solution in turn could benefit from their idea of enforcing hierarchical policies.

Dinda [28] proposed a novel scheme to protect the assets of the grid user against a malicious provider in order to address trust asymmetry. Similar to that proposal, encrypted computation (see, e.g., [29]) offers interesting results for some problems. By performing computations on encrypted data without decrypting it, some tasks can be completed without ever revealing plain text. However, these techniques have limited use outside the domain of some algebraic problems, and their widespread adoption seems unlikely.

7 Conclusion

In this paper, we proposed a protocol for scalable offline attestation based on a grid security architecture that uses virtualization and Trusted Computing technology. Our approach allows the grid user to choose a provider with a trustworthy configuration without interaction, by just selecting an attestation token. The attestation token is published by the provider once and does not have to be generated individually for every potential user. The job submission protocol then ensures that the provider can access the job only in the state considered trustworthy by the user. Current and future work include the implementation of job migration, the support for nodes joining and leaving the grid, and the integration of existing grid infrastructure into the TGA.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International Journal of Supercomputer Applications **15** (2001) 200–222
2. Foster, I., Kesselman, C., Tsudik, G., Tuecke, S.: A security architecture for computational grids. In: Proc. 5th ACM Conference on Computer and Communications Security (1998) 83–92
3. Azzedin, F., Maheswaran, M.: Towards trust-aware resource management in grid computing systems. In: Proc. 2nd IEEE International Symposium on Cluster Computing and the Grid (2002) 452–457
4. Hwang, K., Kwok, Y.K., Song, S., Chen, M.C.Y., Chen, Y., Zhou, R., Lou, X.: GridSec: Trusted grid computing with security bindings and self-defense against network worms and DDoS attacks. In: Proc.

International Workshop on Grid Computing Security and Resource Management 2005. Volume 3516 of Lecture Notes in Computer Science (2005) 187–195

5. Fuggetta, A., Picco, G.P., Vigna, G.: Understanding code mobility. IEEE Transactions on Software Engineering **24** (1998) 342–361
6. Mao, W., Jin, H., Martin, A.: Innovations for grid security from trusted computing. Available online at `http://www.hpl.hp.com/personal/Wenbo_Mao/research/tcgridsec.pdf` (2005)
7. Smith, M., Friese, T., Engel, M., Freisleben, B.: Countering security threats in service-oriented on-demand grid computing using sandboxing and trusted computing techniques. Journal of Parallel and Distributed Computing **66** (2006) 1189–1204
8. Intel Trusted Execution Technology Website: Intel trusted execution technology. `http://www.intel.com/technology/security` (2006)
9. AMD Virtualization Website: Introducing AMD virtualization. `http://www.amd.com/virtualization` (2006)
10. Sailer, R., Jaeger, T., Valdez, E., Caceres, R., Perez, R., Berger, S., Griffin, J.L., van Doorn, L.: Building a MAC-based security architecture for the Xen open-source hypervisor. In: Proc. 21st Annual Computer Security Applications Conference, IEEE Computer Society (2005) 276–285
11. Nabhen, R., Jamhour, E., Maziero, C.: A policy based framework for access control. In: Proc. 5th International Conference on Information and Communications Security (2003) 47–59
12. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: A virtual machine-based platform for trusted computing. In: Proc. 19th ACM Symposium on Operating Systems Principles (2003) 193–206
13. Löhr, H., Ramasamy, H.V., Sadeghi, A.R., Schulz, S., Schunter, M., Stüble, C.: Enhancing grid security using trusted virtualization (extended version). `http://www.prosec.rub.de/publications.html` (2007)
14. TCG Website: TPM Specification version 1.2. Available online at `https://www.trustedcomputinggroup.org/specs/TPM` (2006)
15. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proc. ACM Conference on Computer and Communications Security (2004) 132–145
16. Sadeghi, A.R., Stüble, C.: Property-based attestation for computing platforms: caring about properties, not mechanisms. In: Proc. 2004 New Security Paradigms Workshop (2004) 67–77
17. Rutkowska, J.: Blue pill. Presented at Syscan ’06, `http://theinvisiblethings.blogspot.com/` (2006)
18. Pfitzmann, B., Riordan, J., Stüble, C., Waidner, M., Weber, A.: The PERSEUS system architecture. Technical Report RZ 3335 (#93381), IBM Research (2001)
19. OpenTC Website: The OpenTC project. `http://www.opentc.net` (2006)
20. EMSCB Website: The EMSCB project. `http://www.emscb.org` (2006)
21. Zhao, S., Lo, V., Gauthier-Dickey, C.: Result verification and trust-based scheduling in peer-to-peer grids. In: Proc. 5th IEEE International Conference on P2P Computing (2005) 31–38
22. Cavalcanti, E., Assis, L., Gaudêncio, M., Cirne, W., Brasileiro, F., Novaes, R.: Sandboxing for a free-to-join grid with support for secure site-wide storage area. In: Proc. 1st International Workshop on Virtualization Technology in Distributed Computing (2006)
23. McCune, J.M., Jaeger, T., Berger, S., Cáceres, R., Sailer, R.: Shamon: A system for distributed mandatory access control. In: Proc. 22nd Annual Computer Security Applications Conference (2006) 23–32
24. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture. In: Proc. Annual USENIX Security Symposium, USENIX (2004) 223–238
25. Jaeger, T., Sailer, R., Shankar, U.: PRIMA: policy-reduced integrity measurement architecture. In: Proc. 11th ACM Symposium on Access Control Models and Technologies (2006) 19–28
26. Mao, W., Yan, F., Chen, C.: Daonity—grid security with behaviour conformity from trusted computing. In: Proc. 1st ACM Workshop on Scalable Trusted Computing (2006)
27. Cooper, A., Martin, A.: Trusted delegation for grid computing (2006) Presented at: 2nd Workshop on Advances in Trusted Computing.
28. Dinda, P.A.: Addressing the trust asymmetry problem in grid computing with encrypted computation. In: Proc. 7th Workshop on Languages, Compilers, and Run-Time Support for Scalable Systems (2004) 1–7
29. Algesheimer, J., Cachin, C., Camenisch, J., Karjoth, G.: Cryptographic security for mobile code. Technical Report RZ 3302 (# 93348), IBM Research (2000)