available at www.sciencedirect.com

www.compseconline.com/publications/prodinf.htm

Information Security Technical Report

Daonity – Grid security from two levels of virtualization☆

Haibo Chen[a], *Jieyun Chen*[b], *Wenbo Mao*[c,*], *Fei Yan*[d]

[a]*Fudan University, Department of Physics, Shanghai 200433, China*
[b]*Huazhong University of Science and Technology, Wuhan 430074, China*
[c]*Hewlett–Packard Laboratories China, Beijing 100022, China*
[d]*Wuhan University, Wuhan 430072, China*

Keywords:
Grid computing
Grid security infrastructure (GSI)
Behavior conformity
Trusted computing (TC)
Virtual organization (VO)
Platform attestation
Resource virtualization
Platform virtualization
Secure operating systems

ABSTRACT

The service oriented architecture of grid computing has been thoughtfully engineered to achieve a service level virtualization: not only should a grid be a virtual machine (also known as a virtual organization, VO) of unbounded computational power and storage capacity, but also should the virtual machine be serviceable in all circumstances independent from serviceability of any of its component. At present, a grid VO as a result of service level virtualization only is more or less confined to participants from scientific computing communities, i.e., can have a limited scale. It is widely agreed that for a grid to pool resources of truly unbounded scale, commercial enterprises and in particular server-abundant financial institutions, should also ‘‘go for the grid,’’ i.e., open up their servers for being used by grid VO constructions. We believed that it is today's inadequate strength of the grid security practice that is the major hurdle to prevent commercial organizations from serving and participating the grid.

This article presents the work of Daonity which is our attempt to strengthening grid security. We identify that a security service which we name *behavior conformity* be desirable for grid computing. Behavior conformity for grid computing is an assurance that ad hoc related principals (users, platforms or instruments) forming a grid VO must each act in conformity with the rules for the VO constitution. We apply trusted computing technologies to achieve two levels of virtualization: resource virtualization and platform virtualization. The former is about behavior conformity in a grid VO and the latter, that in an operating system. With these two levels of virtualization working together it is possible to build a grid of truly unbounded scale by VO including servers from commercial organizations.

1. Introduction

Virtualization of resources is the key element in grid computing. Viewed by a user, a computational grid (Bair, 2004; Foster and Kesselman, 1999; Foster et al., 2001) should be a ‘‘virtual machine’’ of unbounded resources. In reality, this virtual machine is ad hoc constructed for the user, comprising a number – possibly very large – of physically separate resources to combine to a federated or collaborated computing environment called *virtual organization* (VO). Fig. 1 illustrates a typical VO structure in a high performance computing setting which comprises of one user (whose platform is in the left),

☆ Work funded by Hewlett–Packard Company. Part of work by Huazhong University of Science and Technology was also supported by the Chinese Natural Science Foundation's ‘‘Research Projects of Great Importance’’ Grant no. 90412010.

* *Corresponding author*. Fax: +86 21 651 04 949.
E-mail addresses: haochen@fudan.edu.cn (H. Chen), Mao_Wenbo@emc.com (W. Mao).

1363-4127/$ – see front matter © 2007 Published by Elsevier Ltd.
doi:10.1016/j.istr.2007.05.005

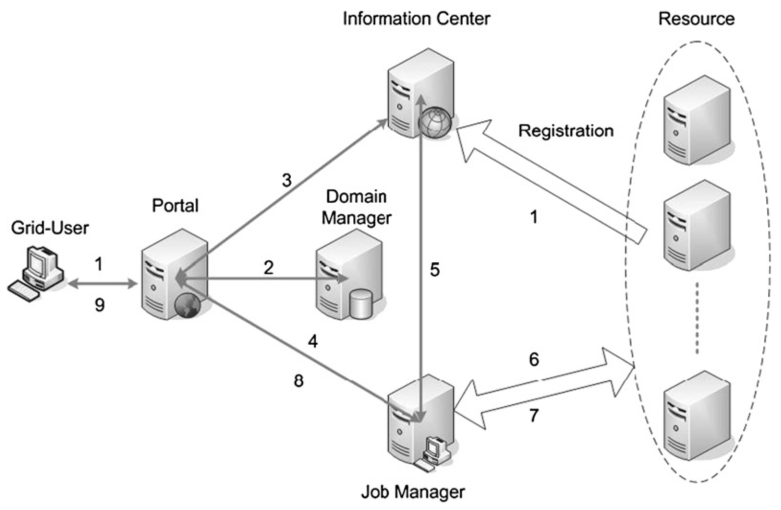

Fig. 1 – A typical grid computing virtual organization.

brokerage servers (Portal, Domain Manager, Information Center and Job Manager in the middle) and job execution farm (resource in the right). This grid VO model depicts typical steps of how grid jobs are submitted and executed. The following enumeration corresponds to the numbered steps in Fig. 1:

1 User begins by logging onto Portal. Independently, resources, which are formed by principals are willing to lease computing/storage resources, register resources to Information Center. The latter will play the role of a resource broker.
2 Domain Manager verifies user's legitimacy of using the grid services. The verification can be based on a prearranged relationship between these two principals (e.g., a credential of user on the basis of a shared secret), or based on a trusted third party's certification (e.g., a credential of user on the basis of a public-key certification). Domain Manager is in a security server's position in a VO.
3 User, having logged onto Portal, can check, by interacting with Information Center, service state and obtain the state information to match his/her jobs' requirement.
4 After obtaining satisfactory service state information, user submits jobs to Job Manager.
5 Job Manager cooperates with Information Center and obtains the addresses of the resources which can satisfy user's job requirements.
6, 7 Job Manager works on user's jobs with selected resources. It supplies user's data to resources, and stores returned computing results for user.
8, 9 Upon completion of jobs execution, Job Manager flags up for user to fetch the results.

The principals in the middle of Fig. 1 (Portal, Domain Manager, Information Center and Job Manager) play important roles to achieve resource virtualization. The organization of these principals in Fig. 1 achieves a service oriented architecture (SOA) for grid computing. A characteristic feature in the SOA can be referred to as a high degree of dependability which covers desirable services collectively in terms of reliability, availability, privacy and scalability. A grid VO can typically execute jobs for a user in a streamlined manner: the user submits a batch of jobs to the middle principals in Fig. 1 in one go and comes back to fetch the result only after the jobs are done. It should be possible that, after the submission of a user's jobs, the jobs must be processed without requiring any further intervention by the user. Moreover, the continuation in the execution of the jobs should not depend on the continuation of serviceability of any component of the VO. We shall see in Section 3 how this property is realized by the leading grid architecture Globus Toolkit Version 4 (GT4). GT4 has several thoughtful designs to virtualize resources with an intension to achieve high dependability for grid computing. Among other means in the architectural design, the security part of the grid architecture plays an enabling role for achieving grid resource virtualization.

However, in Section 3 we will also see through in-depth analyses and discussions a (hidden) point that grid resource virtualization in GT4 is done in a trade-off by working with a weakened notion of trust, even though a strong notion of trust is needed among grid VO participants in order to build a VO of large scale. Our discussions will reveal that the trade-off turns out to be responsible for a limited benefit we can gain from resource virtualization of GT4 as a result of a lowered quality of resource virtualization we have at present.

Therefore the work presented in this article has the following goal: to retain the property of high dependability of, at the same time to strengthen trustworthiness for, the leading grid architecture.

1.1. *Our approach: trusted computing for two levels of virtualization*

We consider that *Trusted Computing Group* (TCG) technologies (Trusted Computing Group; Trusted Computing Group, 2001, 2003; Pearson, 2003) developed by TCG form a practical and readily applicable technical means to serving the need of grid security. TCG is an important industrial initiative for improving computer security by means of a hardware supported security architecture. TCG uses a hardware module *Trusted*

Platform Module (TPM) which is integrated into a computing platform. With an assumed degree of tamper protection on the TPM and its integration to the platform (tamper-evidence is more valid assumption with the server side), TCG in fact assumes a platform owner a potential adversary with respect to policy enforcement in a federated or collaborated computing system. With hardware protected cryptographic capabilities, the TPM integrated into a computing platform is effectively an in-platform trusted third party (TTP) to ensure policy enforcement for all participants, whether the owner of the platform or a guest user. In contrast to the conventional security mechanisms against external, or less privileged, adversaries, the owner of a platform usually is in a privileged position, i.e., a stronger adversary. TCG's goals include to prevent this privileged entity from easy wrongdoing.

Our applications of TCG technologies will involve working on virtualization at two different levels in the grid architecture:

- Resource virtualization in which the work will mainly involve improving virtualization techniques which are used in the present grid middleware.
- Platform virtualization in which the work involves a novel virtualization technique which we propose for strengthening general purpose operating systems upon which a grid middleware run as a trust enhanced application.

1.2. *Organization*

The remainder of this article is organized as follows. In Section 2 we specify assumptions which our solution should be based on, the threat environment for grid computing and security services our solution should provide to collaborated computing. In Section 3 we review advantages and limitations of the current grid security practice. In Sections 4 and 5 we describe why and how TCG technologies can provide a complementary solution to grid security; Section 4 is for strengthening resource virtualization, and Section 5 is for strengthening platform virtualization. In Section 6 we review related work. We conclude in Section 7. We also include in Appendix A rudimental material for TCG technology introduction.

2. Assumptions, threats and security services

2.1. *Assumptions*

We assume that a platform's internal hardware systems (CPU, Memory, internal BUS, etc., and in particular, the TPM) are not in any control and tampering by a wrong hand. By hardware systems being controlled or tampered by a wrong hand, we mean that the hardware system or any component is modified to behave in any way which is not originally designed. However, we do not demand any peripheral hardware systems which are accessed by I/O devices to be trustworthy; these parts are usually shared among operating systems and/or platforms and therefore are vulnerable to unauthorized tampering.

We assume that the software environment over a platform – including the operating system – may be modified and compromised by malicious users, with one exception: a specially tailored code named virtual machine monitor (VMM) (VMware; NGSCB, 2005; Dragovic et al., 2003) is assumed to be the only trustworthy software system.

We consider our trust assumptions to be reasonable because they are supported by the following two insights.

First, it is practical to implement entrust in the VMM code. This piece of software is designed not only to be the most privileged component in a virtualized environment, but also have the following persistent behavior which makes it immune to external intervention not even by a privileged user of the platform. The persistent behavior of the VMM is that it intercepts and monitors all control flows among processes, the operating system's kernel and the underlying hardware systems. In Sections 5.3 and 5.5.1 we will see why the VMM has this privileged behavior. To cause a deviation to this behavior, one needs to modify the code. Then we shall rely on the second insight of protection below.

To prevent unauthorized modification to the VMM code, we call for support from the Trusted Computing Group (TCG) technologies (Trusted Computing Group; Trusted Computing Group, 2001, 2003; Pearson, 2003). The integrity metrics of the VMM code will always be measured by the TCG underlying system upon boot (Sailer et al., 2004b), and we assume the TCG integrity measurement up to the VMM code to be sound. The mechanism of TCG protection on software integrity is given in Appendix A.

The VMM being the only trustworthy software is a much shrunken assumption in comparison to entrusting a whole operating system, and therefore is much more realistic. Under this much shrunken assumption, we consider that the VMM and its underlying TCG foundation forms a trusted computing base (TCB).

2.2. *Threats*

From a user's viewpoint on collaborated computing, the owner of a (remote) platform is a potential adversary who is free to conduct modifications of any software systems, including the VMM running on the platform. The latter case of attack is assumed to be difficult since a successful attack will have to pass the integrity metrics measurement of the TCG technologies and so the user gets a correct integrity report. Considering an infeasibility to succeed a cryptographic attack, tampering the VMM code is essentially not easy than doing the hardware systems.

Having assumed the platform owner being a potential attacker, there is no need for us to differentiate internal and external adversaries. The goal of these adversaries is regarded the same: to attempt a successful deviation from the rule of the game of collaborated computing.

2.3. *Security services*

Conventional security services are in terms of information confidentiality, system integrity, service availability, and commitment accountability. In the work of Daonity, we cover these notions of security services under the unified modifier:

system behavior conformity. Behavior conformity is an assurance that principals (users, platforms or instruments) forming a collaborated computing task must each act in conformity with the rules and policy of the collaborated computing, a policy violation should not be easy. In our goal, the policy of behavior conformity can specify, e.g., that even a platform owner or a privileged entity should not be able read the content in a given memory location on a platform which is even under the full control of the adversary. For another example, the policy can also specify a system integrity protection requirement, in that a malicious operating system should have no way to tamper the executable code and data of a trusted process to cause a deceptive execution result to be returned to a service requester.

2.4. *Non-assumption and non-service*

We make no assumption that a trusted application is semantically error free, i.e., not buggy or even malicious. Instead, we aim to ensure that a malicious application running as trusted processes cannot tamper the execution of other trusted processes.

We also do not guarantee availability of services for and/or from trusted applications. This is because our approach does not protect the whole operating system and thereby some part of the tampered operating system may cause denial of services to a trusted process.

3. Grid security infrastructure – key enabler for resource virtualization

A key technical enabler for resource virtualization and the resultant property of high dependability in grid computing is a result of a thoughtful architectural design which makes servers working in *a proxy-delegation* manner. Any of the servers in the middle and right-hand side in Fig. 1 can be proxy-delegated. For ones working in security (i.e., for most readers of this article), this architecture of server proxy-delegation is best described using figure which is the architecture for Grid Security Infrastructure (GSI) (Open Grid Forum; Foster et al., 1998). Fig. 2 can be thought of as an abstraction of Fig. 1 by highlighting the notion of server proxy-delegation with reasons we now explain.

GSI's VO structure presented in Fig. 2 emphasizes a threat model which considers resource users to attack resource providers (e.g., to spread malware or free ride). In reality mutual authentication is needed and in place, however, the architecture in Fig. 2 has an emphasis on the direction of authentication from the user to the servers. Let us for the moment consider this unbalanced threat-model emphasis on a valid consideration. Then GSI is essentially a result of direct applications of the standard public-key authentication infrastructure (PKI). The VO in Fig. 2 is initiated by a user Alice who is assumed to have an identity certificate issued by a system-wide known grid certification authority (CA). Alice initiates the VO by "recruiting" a member (Proxy 1 in Fig. 2, who is most likely "credential manager" in Fig. 1). Further enlargement of the VO, if necessary, is proxy-authorized to be carried out by Proxy 1 (i.e., without Alice's involvement), and likewise with respect to subsequent "recruitment" of proxies until the VO becomes sufficiently large (e.g., contains $n+1$ members in the case of Fig. 2). The practical meaning of "sufficiently large" should be that the VO contains at least each type of the server entities in the middle and right-hand side of Fig. 1. In order for the process of VO enlargement to be worked out in streamline without tracing back to Alice via a chain of servers (Alice and some of the servers in the chain may have already become off-line and unavailable), GSI applies PKI in the form a chained *proxy certification*: Alice creates a *proxy credential*, which is a public/private key pair; she certifies the public part using her own identity credential and sends the key pair to Proxy 1 who in turn creates a proxy credential and certifies it using the private key (proxy credential) received from Alice; the latter key pair with the certified public part are sent to Proxy 2 (who is recruited by Proxy 1), ..., and so on. This way, a new member can verify, without interaction with Alice, to deem it is indeed Alice who has authorized the organization of the VO. Also, any member in the chain can become off-line while Alice's science can indeed be kept on, as long as recruiting another proxy takes place before a server goes off.

One might want to ask why GSI's VO formation in Fig. 2 (i.e., that of Open Grid Forum) abstracts the grid VO in Fig. 1 into a chained and sequential construction, i.e., why in GSI servers join a VO in a one-after-another manner by sequentially verifying Alice's delegated authorization. It would clearly be more efficient should a joining server down the chain in Fig. 2 contact Alice for her to authenticate it (and authorize it) *directly*. That way there would be no need for the server to verify a possibly very long chain of proxy-authorization signatures which are supported by the long chain of proxy certificates. This very issue was indeed discussed in the

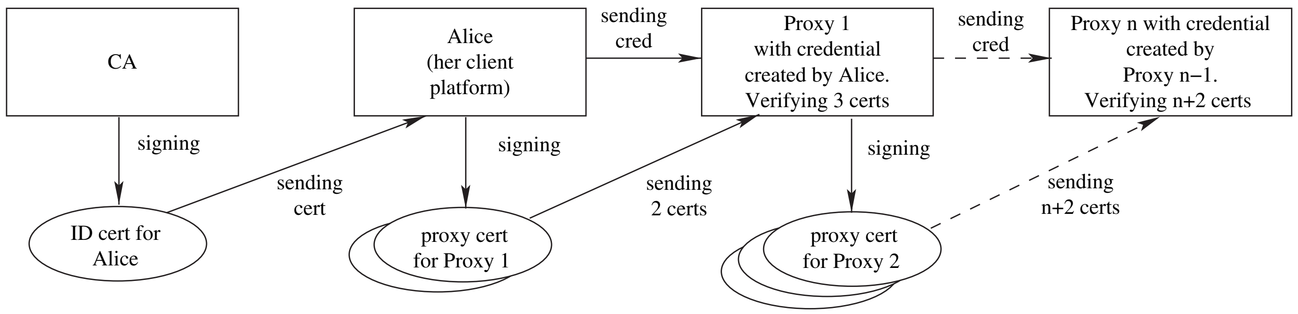

Fig. 2 – VO abstraction in GSI.

original GSI paper under a scalability consideration (Section 5.3 of Foster et al. (1998) "reliance on a single user proxy to forward request to resource proxy lacks scalability"). The key point manifested by the GSI abstraction of the grid VO in Fig. 2 (that of Open Grid Forum) is the following: the chained construction of a VO permits any subsequent joining server to be proxy-authorized by Alice without demanding Alice to be staying on-line in order to materialize the authorization. This way, a VO to be constructed for Alice's jobs can really scale up, e.g., to involve in under-utilized resources in the other side of the planet in a dynamic way. In GSI, this chained delegation of services utilities, in which the user can initiate the delegation of rights without any subsequent intervention, is a featured QoS advantage. This advantageous feature achieves *unattended user authentication* (Open Grid Forum). (Now it is clear why we stated above that Fig. 2 emphasizes user authentication to the servers.) It is obvious that unattended user authentication is a necessary element for achieving grid resource virtualization in a service oriented manner. GSI is therefore regarded as a service oriented architecture.

3.1. *Limitation of GSI: high dependability at the cost of lowered trustworthiness*

With the straightforward PKI application described above, GSI implies a usual and familiar trust model which can be referred to as reputation and best effort maintenance. An unknown entity is deemed trustworthy if it is introduced by a trusted third party (TTP). It is hoped that the introduced entity will behave in a responsible manner by trying its best effort to honor the introduction. We remark that in this introduction based trust model a TTP is usually positioned *outside* the system of partners. In particular, the TTP is usually not placed inside the platforms of the system participants.

Unfortunately, the introduction based trust model actually does not suit well grid security requirements.

For the first reason, it is unclear how Alice can have any control that any of the proxy credentials in the chain will not be misused. Even if it may be reasonable to assume that the servers themselves are trustworthy, it would be unreasonable to assume that they cannot be attacked by an intruder, perhaps a disgruntle operator. Consider, a compromise at a server can be disastrous as a national super computer center can help spreading malware in such an effective way to bring down a grid infrastructure at a grand scale. Actually, GSI does have tried to anticipate this potential threat to some extent. In order to mitigate the potential compromise or misuse of these proxy credentials, GSI stipulates a short lifetime for a proxy credential: 12 h. If we consider this a security policy enforcement realized using a public-key certificate, then this is a rather coarse policy and can greatly limit the power and usefulness of grid computing. To renew the security setting for a VO lasting longer than 12 h is at best a nuisance for the user.

For the second reason, by handing the credential unconditionally over to a next entity down the chain for the full control of rights delegation, we can say that the VO constructed in Fig. 2 is only (or more) suitable for a collegial environment in which partners are colleagues or friends alike. Few financial servers, for example, would be confident enough to allow their under-utilized resources to become part a non-collegial environment as a VO in the GSI architecture. Nowadays, most computational resources are with the much under-utilized servers in financial institutions (with utilization at a mere 10% level, Server Utilization). It is envisioned that to include financial servers into the pool of grid resource providers will be a major boost to the development and deployment of the grid technology.

For yet a third reason, the other direction of threat which is very pertinent to grid security but not possibly preventable by GSI is from a server toward the user. Most grid applications entail code written in one place being executed in another. A host platform's owner should not be able to compromise easily a guest user's security. For example, a guest algorithm running on a host may need protection, in data confidentiality and integrity, for the guest's input to the algorithm and the output result to be returned back to the guest. The protection may need to have a strength against even a privileged entity (e.g., superuser) at the host.

4. Strengthening resource virtualization for grid middleware

We now describe why and how TCG technologies can play a useful role in improving grid security. We assume the reader is familiar with the TCG technologies. Appendix A provides a suitable level of TCG background for the reader to understand the technical terms and their uses in the context of this article.

In Section 4.1 we will introduce the main idea of our solution at an abstract level of description. In Section 4.2 we will explain in detail how to call the standard TCG protocols to realize our solution. In Section 4.3 we will describe the implementation status of this part of the work.

4.1. *Protection of user/VO credential in trusted computing base*

We have discussed in Section 3 that unattended user authentication is a key technical enabler for grid resource virtualization. GSI achieves unattended user authentication by a VO participant handing a proxy credential (i.e., a cryptographic key) to another VO participant. In order for the VO construction to be done automatically, the proxy credential is simply stored in the file system of the platform protected under the access control of the operating system. The obvious danger of leaving a private key in the file space is mitigated by stipulating a short lifetime for the proxy certificate. The default lifetime of a proxy certificate in the GSI is 12 h. Upon expiration, a new proxy certificate must be re-issued. We feel this is an unacceptable security exposure.

Assuming that with TCG technologies becoming more and more pervasive,[1] we consider to store the VO credential inside the trusted computing base (TCB) of the participating platforms. Fig. 3 illustrates the creation and enlargement of a VO using TCBs to protect the VO credential. This VO has

[1] According to IDC's forecast (Rau, 2006), by 2009, about 80% of ×86 platforms which include desktop PCs, mobile PCs and ×86 servers will be equipped with TPMs.

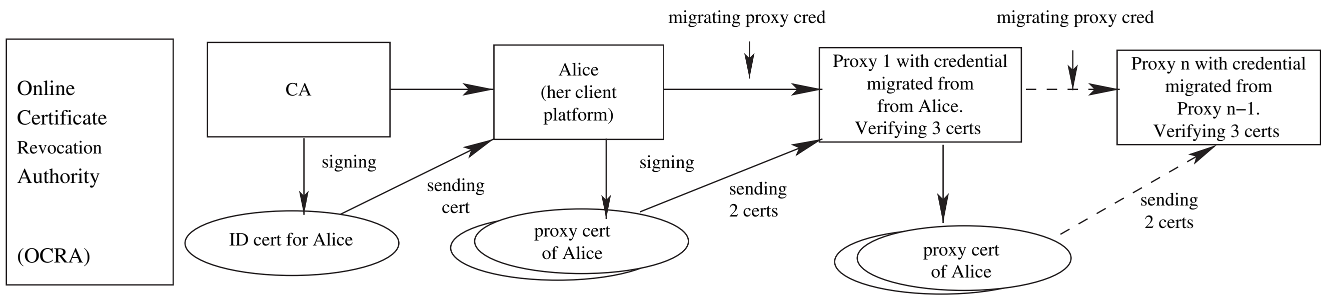
Online Certificate Revocation Authority (OCRA)
CA
signing
ID cert for Alice
sending cert
Alice (her client platform)
signing
proxy cert of Alice
migrating proxy cred
sending 2 certs
Proxy 1 with credential migrated from from Alice. Verifying 3 certs
proxy cert of Alice
migrating proxy cred
sending 2 certs
Proxy n with credential migrated from Proxy n−1. Verifying 3 certs

Fig. 3 – VO construction in Daonity using TCG credential migration.

essentially the same structure as the GSI architecture in Fig. 2, and hence Daonity will retain the property of unattended user authentication of GSI. The only difference is that we now can use TPM to hold the VO credential and have it *migrated* within the TPMs of the VO platforms. *Credential migration* is a TCG function designed for a user to retain security environment when changing (e.g., upgrading) platforms. The essence of Daonity is to make an extensive use of this standard TCG functionality.

Apart from using an ''Online Certificate Revocation Authority'' (its role to be described in Section 4.2), the construction of this VO has no essential difference from the GSI VO in Fig. 2. The VO's creation is also initiated by Alice, and each enlargement step proxy-authorized to a proxy without interacting back to Alice. Now with the involvement of credential migration, the use of chained proxy certificates is avoided. A single proxy credential, which we shall name ''private-key-VO,'' will be created in the TPM of Alice's platform and proxy-authorized to migrate to each of the TPMs of the principals participating the VO. The matching public-key-VO is certified by Alice. With the VO credential in TPMs, the coarse policy of 12 h VO lifetime in GSI is now unnecessary.

Grid computing certainly needs security policy enforcement at the VO level. However, it seems that GSI has quite limited means to do so. Like the case of stipulating 12-h lifetime for a proxy credential, security policy enforcement in GSI is mainly in the form of a pre-setting statement, perhaps specified inside a certificate. We feel this method is not a very powerful one. We envision that the use of TCG technologies in grid security will result in a major improvement on the method of policy enforcement in grid computing. Let us now provide more details.

4.2. *Policy enforcement afar*

With a conformed behavior enforced by a TPM as an in-platform TTP, we can further realize policy enforcement in a VO. A simple use case scenario we can describe here is that a member who is removed from the VO shall not be able to take any VO data away. In this use case, a TPM enforces a conformed way of using the VO credential: the credential is usable in each instance only after the TPM has obtained an explicit approval from the ''Online Certificate Revocation Authority'' (OCRA). The OCRA can be instructed by Alice to enlist the revocation of the VO membership with a given TPM. Once a revocation of membership with a TPM is enlisted in OCRA, the TPM will no longer use the credential anymore. This way, Alice can remove a member from the VO without letting it take away VO data. Notice that this removal procedure is non-interactive: no need to obtain the consent from the member to be removed. Non-interactive removal prevents possible refusal and so the VO functions cannot be stopped by a non-cooperative party.

One might want to ask: how can a TPM be so clever to enforce complex policy afar? The crux rests in the collaboration between the TPM and the trusted software stack the TPM serves. In the TPM there are a rack of *Platform Configuration Registers* (PCRs). As TCG's ''*Root of Trust for Measurement (RTM)*'' function, a PCR records, via ''eavesdropping'' or ''wiretapping'' the input–output bus (aka ''low-pin count bus'' in the ''southbridge'' of an $\times 86$ platform) between external devices and the CPU's input–output control hub, the hash total value of the software executables which have been loaded from the external drives to the platform so far. To work with this RTM function, the TPM is programmed to have the following simple behavior: to perform a critical service (e.g., decryption or digitally signing) upon the current PCR value in the TPM (this is the ''eavesdropped'' software environment of the platform outside the TPM) matching that in the service description given by Alice. The PCR value which is ''eavesdropped'' by a remote TPM can be reported to Alice via *attestation*, i.e., Alice can deem the trustworthiness of the software environment in a remote platform (''*Root of Trust for Reporting (RTR)*''). Thus, a critical service from a TPM to the external software environment can be made only available to the software environment when the software environment has been approved by Alice. In our simple use case scenario, for example, the software environment is one which includes a program to check the OCRA for whether or not Alice has revoked the VO membership of the underlying TPM.

To detail the know-how, we first introduce a protected functionality inside the TPM which permit the TPM to take control whether or not to provide a critical service to the external software environment.

4.2.1. *To serve or not to serve? An internal decision by the TPM*

A TPM contains a number of *Platform Configuration Registers* (PCRs). Each PCR is a 20-byte register of volatile memory. The use of a PCR is to record the measurement result on the software environment which has been loaded onto a platform. A PCR records the software measurement in the following "extension" formula:

$$\text{PCR} \leftarrow \text{SHA}-1(\text{PCR}\|\text{a software binary executable}). \quad (1)$$

A recorded measurement on the software environment over a platform can be *attested* to a remote querier using a standard challenge-response mechanism (in cryptographic protocols). Under the assumption that the TPM does not cheat a querier, the latter can re-compute the hash total value to deem if the remote platform has loaded a needed software environment desired by the querier.

A *protected cryptographic capability* is a basic function of a TPM. With a crypto key and algorithm inside the TPM, basic crypto operations such as decryption and signature creation can be conducted in a protected manner without revealing the key to the external environment. More importantly, a TCG complying TPM can be instructed by a remote stakeholder only to provide a crypto operation if the external software environment is deemed desirable by the remote stakeholder. Let Alice be a stakeholder remote to Bob's platform. Alice can create the following blob (formulated in pseudo code) for Bob's TPM to process or not to process, depending on whether Bob's platform has a software environment measurement Alice approves:

```
Encrypt(Bob-TPM-PubKey, Data, PCR)
```

In this line of pseudo code, "`Encrypt`" is a public-key encryption algorithm, "`Data`" is the information encrypted inside the blob, "`PCR`" is the PCR value which Alice wants to instruct the TPM of Bob that decryption of "`Data`" should only be served if the local PCR value in the TPM matches that in the blob. In order for Bob's TPM to decrypt this blob, the two PCR values must match, i.e., Bob needs to load his platform with the software environment which Alice approves. In the actual use of this formula, Alice can provide a list of possible PCR values which she accepts; Bob's TPM will serve if the local PCR value in the TPM meets one in the list. This makes it possible for the situations Alice and Bob using different platforms, or Alice permits varied sequences of software loading at Bob's platform.

To exemplify the usefulness of this conditional decryption service by a TPM, we can imagine that the software environment approved by Alice should include an executable which has the following behavior: it checks a revocation authority regarding the certificate of "`Bob-TPM-PubKey`" before loading the decryption result to the memory. If Alice is able to instruct an authority to revoke the certificate, then the blob of encrypted "`Data`" becomes not usable to Bob anymore. It is now not difficult to imagine a rich way of policy enforcement afar.

This TCG mechanism of protected cryptographic capability is functioning even for the complying TPMs under the TCG TPM specification version 1.1b. It has also been tested working properly for the TCG TPM specification version 1.2.

4.2.2. *Realization detail*

In TCG's specification, there are two modes of credential migration. An implementor can choose a mode by specifying one of the following parameters in the function/protocol calls:

`TCPA_MS_MIGRATE`: This mode is in the real sense of credential migration because it is designed for backing up a credential from one TPM to another. We use this mode in our implementation of Daonity since in our application we need to move a VO credential from one platform to another.

`TCPA_MS_REWRAP`: This mode is designed for moving a credential which is already secured under a parent key to be re-secured under a new parent key. This move can include the case that the credential remains under the same storage root key, i.e., secured by the same TPM. While this mode may also permit a credential to move from one SRK tree to another, i.e., between different TPMs, doing so involves more complex protocol calls than the other mode does. (This mode is much simpler if the move is within the same SRK tree.)

We note that in both modes, the credential can be still used (loaded) under the original SRK system after a migration. This means that a migration of a credential can result in a sharing of the credential by two SRK systems or two TPMs. This is a desirable result we want in order to build a grid VO sharing a VO credential.

Below we provide annotated pseudo-code TCG protocols and function calls to describe how we have really achieved migration of a VO credential between two platforms. Generalization of these protocols and function calls to a larger VO is standard. We notice that for exposition clarity, we have omitted many parameters and specification details (such as `TCPA_MS_MIGRATE`) from our pseudo-code presentation.

To migrate a VO key from A to B (here and below A and B stand for TPM-A and TPM-B), a series of protocols will run between A and B.

4.2.2.1. *Attestation of B's software environment to A.*

The first protocol is for B to attest to A the status of B's software environment measurement. Let the attestation output to A include the following two elements:

- PCR Value `PcrValueB` which is constructed inside the TPM of B using the "PCR-extension" formula (1) where "a software binary executable" in (1) is the executable loaded onto B sequentially when B boots its software environment. `PcrValueB` denotes the final value of PCR resulted from the "PCR-extension" formula (1).
- Public-key `B-PubKey`. This is a public key created inside the TPM of B under the software environment measured as `PcrValueB`. Its role is to receive a VO credential to be migrated to B. In a moment we shall see more detail on how B has created `B-PubKey`.

4.2.2.2. *Preparation of a PCR-conditioned VO credential by A.*

After the attestation, A can set its PCR-No 8 to `PcrValueB`, whose length is 20 bytes, using the following command (here and below the use of the 8th PCR is for illustration purpose):

```
SetPcrValue(PcrComposite, 8, 20, PcrValueB)
```

A then creates a VO key named `VOKeyPair`. This is an RSA Key pair of which the private part will be migrated to B. This key pair is created using the following command:

```
CreateKey(VOKeyPair, A-SRK, PcrComposite)
```

Here, A secures the private part of `VOKeyPair` under its storage root key A-SRK. This VO key is created to be bound to PCR-No 8 of the value `PcrValueB`. This means that any instance of use of the private part of `VOKeyPair` by A is conditioned that the value of PCR-No 8 in A should be `PcrValueB`; otherwise, loading this key in A will fail with returning an error message.

4.2.2.3. *Creation of a migration blob by A*. Next, A can create a migration blob to send to B. The first step is to authorize `B-PubKey` which has been output to A in the attestation protocol (see above). The authorization of `B-PubKey` is done by generating a "migration ticket" as follows:

```
AuthorizeMigTkt(B-PubKey, &TktLen, &MigTkt)
```

Now A creates a migration blob as follows:

```
CreateMigBlob(VOKeyPair, B-PubKey, TktLen, MigTkt,
&MigBlobLen, &MigBlob)
```

In this function call, A encrypts the private part of `VOKeyPair` using `B-PubKey` which has authorized by A (in the ticket) earlier. The protocol will output to B the following migration blob data: `(MigBlobLen, MigBlob)`.

Below let us look at what B should do in these protocols.

First, we recall that in the time of running with A the attestation protocol, B has created the key pair `B-PubKey` under the environment measured as `PcrValueB`. The private part of `B-PubKey` is encrypted in a secure storage rooted by B's storage root key `B-SRK`.

Having received from A the migration blob data `(MigBlobLen, MigBlob)`, B can convert it as follows:

```
ConvertMigBlob(B-PubKey, MigBlobLen, MigBlob)
```

In this function call, the migration blob is decrypted by the private part of `B-PubKey`, and as a result, the VO credential `VOKeyPair` becomes a son of `B-PubKey` in B's secure storage.

Because `B-PubKey` has been created to be usable conditioned on the software environment measured by the given PCR value, the following LoadKey function call will only work if the correct PCR value is in B:

```
LoadKey(VOKeyPair, B-SRK)
```

In TCG's specification, if PCR value is incorrect, the above function `LoadKey` should fail. In our experiments, if the PCR value is incorrect, `LoadKey` does not return error immediately, instead, an error will occur later when we call `Unbind` function (see below).

4.2.2.4. *VO confidential communications*. Suppose that the migration is successful, the subsequent VO confidential discussions can be encrypted using the public key of `VOKeyPair`. Both A and B know that the decryption by any VO member must be conditioned in a software environment setting which is measured as a PCR value in `PcrValueB`. The encryption step is as follows:

```
Bind(VOKeyPair, BindDataLen, DataToBind)
```

and the corresponding decryption step is

```
Unbind(VOKeyPair, BindDataLen, DataToBind, &UnbindData)
```

Here, `UnbindData` is the plaintext output.

As we have mentioned above, `Unbind` will fail to decrypt if the `LoadKey` function has been called when inconsistent PCR values were used.

4.3. *Implementation status*

Planned as a contribution to leading grid standard, Globus Toolkit, which has been developed in open source, our work will also be in open source as a component for the ever evolving Globus Toolkit. In the time of writing this article we have completed the design, specification and implementation for a proof-of-concept (PoC) system of the Daonity system and have released it.

The PoC system works on PCR values loaded in two TPMs, one is migrator (the TPM with an out-going VO credential), and the other, migratee (the TPM with an in-coming VO credential). We have assumed that the migrator and migratee TPMs share the same PCR-No 8. We have tested the workability: with the same PCR value in the two TPMs, the migration will succeed; otherwise, it fails. However, in reality, sharing of a PCR value between two TPMs as an assumption does not make a useful sense. We need an attestation protocol, e.g., one described in Section 4.2.2, to allow the migratee to report its PCR value (i.e., the software environment outside the migratee) to the migrator for the latter to verify the acceptability. Such an attestation protocol has not been implemented in the first release of the PoC. Implementation of the attestation protocol will be the immediate next step of the work.

The implementation of the Daonity PoC system has been greatly benefited from the open-source Trusted Software Stack (TSS) package TrouSerS, and the open-source grid middleware package Globus Toolkit (GT4). In fact, apart from the TPM migration component which we have described in Section 4.2.2, all other TSS parts of Daonity are readily adapted and modified from TrouSerS, and plugged into GT4. With TCG's TSS soon to become available for TPMs of all complying vendors, we have planned to add the migration part to TCG's TSS and so Daonity will become usable over TPMs of those vendors.

5. Strengthening operating systems via platform virtualization

The conformed behavior for the grid middleware we have been working on up to this end is valid only under an assumption that the operating system upon which the grid middleware runs will not provide an attacker with an easy shortcut. No matter how strong the policy enforcement in the middleware is, if an attacker can exploit weaknesses in the operating system then the policy can still be violated easily. For example, the key management scheme which realizes TCG's Root of Trust for Storage (RTS, see Appendix A.1.3) have most part of the cryptographic operations executed outside the TPM; namely, a session key can actually be extracted from a register or a memory location if the operating system permits the extraction (a commodity OS usual does). It is vitally important that an operating system upon which trusted applications run should not be easily tampered with for policy deviation, not even by privileged user such as the platform owner. Thus, our work inevitably involves strengthening the operating system.

We shall take a platform virtualization approach to strengthening commodity versions of the Linux OSes. This part of the work is named CHAOS – Conformity and High Assurance within OSes. The work reported here is organized as follows. In Section 5.1 we overview contemporary operating systems on their unsuitability to serve a trustworthy execution environment for applications which need behavior conformity. In Section 5.2 we specify requirements related to our work. In Section 5.3 we describe the working principle of virtual machine monitor and how it forms a confined and practical trusted computing base for our use. In Section 5.4 we provide an overview on the CHAOS architecture. Finally in Section 5.5 we describe realization details of CHAOS.

5.1. *Security overview of commodity operating systems*

So far standard grid middlewares run on commodity operating systems (e.g., GT4 runs on Linux). Unfortunately, contemporary commodity operating systems are essentially untrustworthy. This point can be discussed in three aspects.

Firstly, these OSes are usually very big, complex – meaning a non-negligible probability for them to be buggy – and developed without *a priori* and special considerations on security over other prioritized properties such as optimization for performance, resource utilization and other useful functionalities such as flexibility and openness. This is nevertheless the least problematic part when we talk about OS security because errors as a result of problems in this aspect are most likely non-intentional and non-maliciously designed.

Secondly, the design principle of contemporary commodity OSes actually violates the principle of least privilege (PoPL) National Computer Security Center, 1991. The PoLP requires that only least privilege should be granted for a process to complete a task. Instead, commodity OSes usually expose permissive interface in terms of system calls to applications thereon. As a consequence, applications with diverse security requirements are poorly isolated within an operating system. A success in tampering one application will likely enable easy ways to further tampering other applications in the same operating system, and so the weakness can be exploited by a malicious user to gain an attacking advantage.

Finally, these OSes provide poor guarantees on the execution integrity of software systems, in particular if the execution environment is in remote setting. Since a machine owner is usually granted with unrestricted permissions, a service requester can hardly have confidence that the code and data run and processed remotely will not be used in some unauthorized way, e.g., confidential information to be disclosed to a wrong hand, and proprietary material to be made available to a competitor. Meanwhile, these OSes lack a confident means for a remote user to verify that results returned from a collaborated computing application is tamper-free.

5.2. *Requirements*

We aim to create a tamper-resistant execution environment for security sensitive grid applications by applying TCG technologies to hardening the OS. We are aware of several desirable requirements in such a system.

- It must retain full functionalities as existing operating systems, thus applications will not sacrifice functionalities when running on it.
- It should retain backward compatibility so that existing applications need not be reconstructed to run on the platform.
- It should be cost-effective that commodity hardware can be used as building blocks to construct it.

5.3. *Virtual machine technology – virtual machine monitor*

Making an operating system "trustworthy" proves to be a non-trivial task. On the one hand, to talk about trust in the context of a commodity operating system seems not very meaningful. On the other hand, to design and realize a new OS from scratch with trust and security as *a priori* design considerations should be a grandiose task beyond the scope of our current work.

We take a practical approach to "hardening" a commodity OS by working on virtual machine monitor (VMM) techniques (VMware; NGSCB, 2005; Dragovic et al., 2003). A VMM is a layer of the most privileged code which has been abstracted from the rest of the OS. In the virtual machine architecture, this code is placed in the layer under the rest of the OS and specializes the functions of intercepting, monitoring and processing service calls from processes to the OS and the underlying hardware systems. For example, a cut-and-paste action from one process (e.g., a web page viewing application) to another (e.g., a document editor) will cause the following events in the OS: it first issues a service call for a read access to a memory location used by the former process, it then issues another call for a write access to a memory location of the latter process. If this OS is on top of a VMM, then these calls will all be first intercepted by the VMM to be relayed to and from the hardware layer under. A VMM can serve a plural number of (even different) OSes to use one piece of hardware resources. To this end we can see that, in essence, the VMM acts as a middle man in between not only processes but also a process and the OS kernel. *All* inter-process communications and inter-process-OS communications will be intercepted by the VMM.

The above behavior of the VMM is *privileged* and *mandatory*. A malicious entity, even controlling the OS kernel, cannot cause a deviation for the interception behavior of the VMM to be bypassed, unless the VMM code is modified by the attacker (then see our countermeasure from TCG technologies to be described in a moment). A malicious OS kernel can refuse to cooperate with the VMM, and this seems to be the only feasible attack which is of course not very interesting in collaborated computing applications. This property of non-deviation is essential for achieving a meaningful sense of entrusting in the system's behavior. In our use of the VMM, we focus on the following functions:

- interception of inter-process and inter-process-OS communications;
- memory location isolation;
- sealing of CPU context, memory contents and I/O data which are owned by an application which needs behavior conformity as a security service.

With the VMM having the above mandatory property of intercepting inter-process communications, we can consider that the VMM form a trusted part of the system. By a trusted VMM, we mean that the code is to be measured by the TCG mechanism of the Root of Trust for Measurement (RTM, see Appendix A.1.2). For applications needing behavior conformity, the participant of a collaborated computing will make an authenticated boot of the genuine version of the VMM. The fact of the authenticated boot of the VMM can be attested to a remote participant of the collaborated computing (see Appendix A.1.4) as evidence of no tampering on the boot sequence and the correct VMM running on the platform participating the collaborated computing. This prevents installing a root-kit below the VMM or launching a malicious VMM.

Thus, the TCG system and the integrity protected VMM form the trusted computing base (TCB) for our solution.

Another important property of the VMM layer is *transparency*. Applications which can run on an OS without the VMM layer can also run on an OS with the layer. This property enables to a commodity operating system to work with the VMM, and thus to retain the existing programming interface and functionalities. Thus our goals of backward compatibility are satisfied.

5.4. *CHAOS architecture overview*

Fig. 4 depicts the system architecture of CHAOS. This system design is adapted from that of the open-source VMM system Xen (Barham et al., 2003). While Xen mainly focuses on efficient utilization of platform hardware resources by multiple VM OSes, our adaption in CHAOS has a focus on monitoring security sensitive applications which are as VMs. In CHAOS, security sensitive applications are executed as *trusted processes*, which should be resilient to tampering by other processes and even the OS kernel. This is a result of protecting the CHAOS VMM by TCG technologies. CHAOS transparently creates a tamper-resistant environment by implementing a *trust management layer* in the VMM.

As shown in the architecture of Fig. 4 for a general virtualization scenario, there could be several untrusted operating systems running simultaneously atop the VMM. Note,

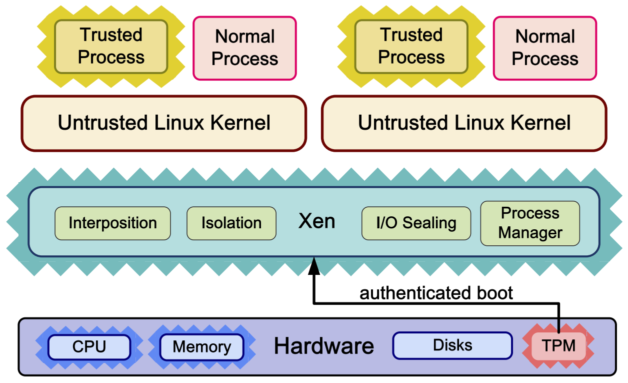

Fig. 4 – The CHAOS architecture: components surrounded with zigzag are protected.

although a management VM (not shown in the figure) in CHAOS contains tools to create and manage other VMs, that VM is still restricted by the VMM from tampering the executions of trusted processes. Examples of tampering are, e.g., mapping the process owned memory location and getting the execution context.

5.5. *CHAOS realization*

We have implemented a prototype system of CHAOS for Linux 2.6.16 running on Xen 3.0.2. The hardware platform is ×86 architecture. However, we believe that the architecture and approach of CHAOS are not Xen-specific and its concept and architecture should be applicable to other VMMs and operating systems.

Untrusted processes and trusted processes co-exist atop an untrusted OS kernel. The VMM only monitors trusted processes without interventions to normal process. For a trusted process, the protection of sensitive information needs to be considered from three categories: in CPU execution context, in memory pages and in I/O data. In the remainder of this section, we provide an abstract level of descriptions of the CHAOS implementation work with respect to these three categories of protection. The respective protection measures are *interposition*, *isolation* and *I/O sealing*. The following three subsections (Sections 5.5.1–5.5.3) describe these measures respectively.

5.5.1. *Interposition*

The interposition mechanism intercepts all kernel-user transitions to give the VMM chance to protect sensitive information.

Fig. 5 depicts the control flow transitions for a *trusted system call* (TSC) and a normal system call in CHAOS. Normally, a system call is essentially an interrupt[2] (0 × 80 in Linux) intercepted and forwarded by the VMM. However, for the sake of performance, Xen optimizes the system call forwarding and allows user processes to directly call into the operating system kernel. To intercept control transitions incurred by system calls, CHAOS needs to intercept system calls made by a trusted process. It is rather straightforward to make Xen to regain

[2] Other forms such as *sysenter* (Intel) and *syscall* (AMD) can be similarly handled in CHAOS.

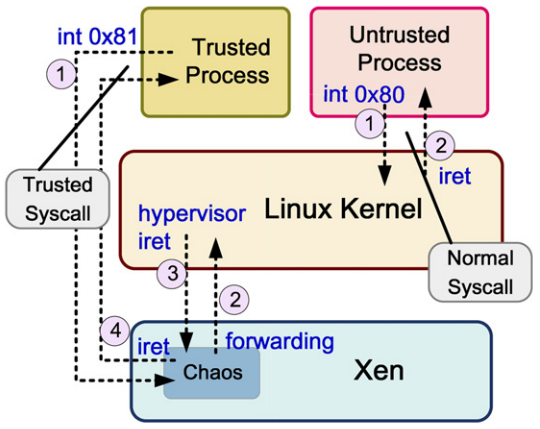

Fig. 5 – System call control flow for trusted system calls and normal ones in CHAOS.

control over *trusted system calls* (*TSC*), simply by replacing the system call entry in the *interrupt description table* (IDT) with a routine provided by CHAOS. The routine performs necessary operations such as isolation and sealing to protect sensitive data. Moreover, to avoid the performance loss for normal processes, CHAOS utilizes another unused interrupt line (i.e. 0×81) for trusted system calls. We provide a binary rewriting tool to rewrite all system calls in a trusted application to utilize the new interrupt line, i.e., to change the system call code from *int* 0×80 to *int* 0×81.

To protect CPU context, the VMM interposes all the transition between user space and kernel space. Upon the interception of a transition, the VMM is responsible for saving and restoring the CPU context owned by a trusted process. The VMM also conceals some general purpose registers from the OS kernel. No replay attack is possible since the OS kernel cannot set a malicious CPU context that resumes the execution of a trusted process in a previous execution.

5.5.2. *Memory isolation*

The isolation mechanism isolates the CPU context and memory location owned by a trusted process from accessing by the OS kernel.

The memory owned by a trusted process is protected in two ways. First, the VMM hides the user-level mapping in a page table to prevent the OS kernel from accessing the memory owned by a trusted process. Second, the VMM tracks the usage of each memory page to prevent unauthorized mappings to memory pages owned by a trusted process. Consequently, the OS kernel cannot tamper the code or inspect the data for a trusted process. Notice that hiding the user-level mapping will prevent the process-kernel data exchanges. To handle this, recall that the VMM acts as a middle man to assist the data exchanges by copying the data between the OS kernel and a trusted process.

5.5.3. *I/O sealing*

The I/O sealing mechanism transparently encrypts and decrypts sensitive I/O data to prevent the OS kernel from observing the data.

Sensitive data of a trusted process, when to be input from or output to persistent storage, will be protected by cryptographic means. CHAOS transparently encrypts sensitive I/O data in the VMM layer. Generally, I/O operations are made using system calls or memory-mapped I/O. For system calls, the VMM intercepts each I/O related system call and encrypts the data before passing it to the OS kernel. On fetching it from the OS kernel, the VMM will also decrypt the data before passing it to the user space. For memory-mapped I/O, the VMM intercepts the page table updating requests and decrypts the data on the first page fault. When the page is unmapped (i.e., cleared the mapping from the user-level page table), the page is encrypted again to prevent the OS kernel from observation.

Another important issue is to secure launch grid applications. To prevent information leakage during the loading process, CHAOS requires the program code and data to be encrypted using a key provided by the platform. The VMM will assist the process creation to decrypt the code and data. Because only the VMM can decrypt the code and data during process creation, there is no leakage of sensitive information during this process.

6. Related work

We now review related work. This is separated into two parts with respect to our work on two levels of virtualization: those for enhancing security for collaborated computing above operating systems, and those for enhancing operating systems trustworthiness.

6.1. *Security for collaborated computing above operating systems*

SHEMP (Secure Hardware Enhancement for MyProxy) (Marchesini and Smith, 2005) is a system which hardens MyProxy, the on-line PKI credential management servers, using a hardware trusted computing base. A MyProxy server as an important server is an attractive target to attack. Strong hardware based protection on the credentials and user passwords which are centrally managed by a MyProxy server is very desirable. The hardware TCB proposed by SHEMP is an IBM cryptographic co-processor. A hardware protection method for credential is also proposed in Lorch et al. (2004). These proposed just use a TCB secure hardware as secure storage devices. The protection is considered on local resources and assets. By contrast, in Daonity, we treat TPM as a TTP in remote platform, here we intend to enforce security policy afar, i.e., in addition to care about local resources, Daonity also cares about protection of resources afar.

Property-based attestation for computing platforms (Sadeghi and Stüble, 2004; Chen et al., 2006) argues that the attestation functionality proposed by the existing specification of the TCG can be misused to discriminate certain platforms and the operating systems running on the platform. Therefore the really essential element for an attestation should be properties of the software systems in the platform, not the software systems themselves. How to make use of property-based attestation may be of an interest for a future work of Daonity.

Apart from TCG for grid security, many other works have also applied TCG technologies to applications with remote access control (Sailer et al., 2004a, 2005a,b; Haldar et al., 2004; Sandhu and Zhang, 2005) in distributed computing. The work of Sailer et al. (2004a) and Sailer et al. (2005b) proposed a remote access control system for client-server based systems using integrity measurement and software stack measurements. The work of Haldar et al. (2004) employed language level virtual machines to attest the behavior of remote entities. Compared with Daonity, it seems that none of these systems support task delegation and secure redistribution of tasks, as they do not involve credential migration. The approach in Sandhu and Zhang (2005) is similar to Daonity in that it also supports credential migration and secure redistribution. However, it is mainly for policy enforcement in Peer-to-Peer computing.

Architectural supports for software copy and tamper resistant have been intensively studied in Lie et al. (2000), Shi et al. (2004), Suh et al. (2003a,b). These architectures use secure processors to encrypt/decrypt data when interacting with off-chip memory. These technologies can provide very strong protection to OS kernel and applications. With the combination of a kind of remote attestation mechanism such as using TPMs, secure processors can serve well for trusted grid computing.

6.2. *Previous efforts on OS trustworthiness*

To guarantee operating systems level security, Terra (Garfinkel et al., 2003) first advocated to build a trusted platform upon a trusted virtual machine monitor (TVMM), instead of directly using commodity operating systems. TVMM isolates the upper VMs to allow a program to run in an authenticated VM. NGSCB (2005) proposed to use a similar verifiable micro-kernel as trusted base and provide trusted services for secure applications running thereon. Machine partitioning (e.g. Terra; Garfinkel et al., 2003; NGSCB, 2005; Peinado et al., 2004, 2005) solved this problem by multiplexing commodity operating systems and dedicated operating systems on a single hardware platform. While they could result in strong isolation between applications with diverse security requirements, they do not essentially resolve the impasse. Secure applications can only execute in dedicated operating systems, which provide only restricted functionalities and require a reconstruction of applications. sHype (Sailer et al., 2005a) focuses on providing a secure resource sharing between VMs. Sharing of resources is controlled by a MAC (Mandatory Access Control) module. Currently, Daonity aims at providing an abstract model for grid security and focusing on application level security. Therefore, Daonity is orthogonal to these OS-level approaches. No doubt, these OS enhancement approaches will complement Daonity and further enhance grid security under secure infrastructure secured by a Daonity-like solution.

Micro-kernel based approaches (Chen and Morris, 2003; Shieh et al., 2005; Härtig et al., 2005) are promising in reducing the trusted computing base. However, they either require a redesign of operating systems and applications (Shieh et al., 2005), or they only provide restricted functionalities for secure applications (Chen and Morris, 2003; Härtig et al., 2005).

Recent architectural enhancements (Lie et al., 2000; Suh et al., 2003a) aimed to provide a private, tamper-resistant execution environment for high-assurance applications. XOMOS (Lie et al., 2003) examined these enhancements to support trusted processes running in an untrusted operating system. However, as the brought benefits greatly surpass the incurred overhead in terms of cost, complexity and performance, they are inherently expensive and impractical to be adopted by industry.

Mandatory access control (MAC) systems have been widely used to enhance system security. One typical system is SELinux (Loscocco and Smalley, 2001), which integrates fine-grained access control policies to control access to many system resources in Linux. However, the policies are usually rather big and complex, which are hard to derive and to verify the completeness. Eros (Shapiro et al., 1999) is a micro-kernel based capability system that integrates MAC to control accesses to critical resources. Asbestos (Efstathopoulos et al., 2005) and Histar (Zeldovich et al., 2006) restrict the privilege of processes by using capability-like mechanisms to explicitly track and restrict the process privilege. By contrast with CHAOS, they trust the operating systems and machine owners, thus have a different trust model. Moreover, Eros, Asbestos and Histar are not designed to retain backwards compatibility. They build new operating systems from scratch and require existing applications to be ported to run in them.

Recently, a method (Jaeger et al., 2003) on how to realize information flow model on SELinux was proposed to isolate multiple security layer processes and sensitive data in memories for different processes. PRIMA (Jaegar et al., 2006) was proposed to provide TCG-based dynamic integrity measurement based on CW-Lite mode (Shankar et al., 2006). They can be useful guiding our future work in this direction.

System virtualization has gained great popularity in grid computing (Figueiredo et al., 2003; Krsul et al., 2004; Adabala et al., 2005) due to the fact that virtualization is promising to provide a unified management of heterogeneous grid resources with an enhanced security. We believe the approach in Daonity could be seamless integrated to these systems to enhance their security in the form of policy enforcement afar.

7. Conclusion

As grid security is becoming a more and more important topic, a number of problems remain un-tackled by the existing grid security solutions. We have identified that policy enforcement afar is an essential requirement for grid security, or in fact, for any distributed computing applications where a partner-and-adversary threat model applies. We have argued that trusted computing technology, thanks to its inherent property of behavior conformity, can provide suitable solutions to the identified problems in the existing grid security solutions.

As hardware and software supports for TCG technologies are gradually becoming widely deployed, it is timely to consider how such tools can be used to maximum effect in enhancing trust and security in grid environments. The work of Daonity can be regarded as an early trial. The policy-enforcement-afar property in the work of Daonity is still in a very primitive stage, mainly and specifically, to limit the behavior of the TPM owner. Nevertheless, sufficient innovations have been identified through the progress of the work.

While there are many systems and innovations in creating tamper-resistant execution environment, our work in both levels of virtualization differs from previous efforts in that it relies on existing programming languages, conventional operating systems and commodity hardware to retain backward compatibility with existing applications. It is designed to require only minimal changes to commodity operating system and to incur only modest performance overhead to applications demanding trustworthiness.

Service oriented architecture (SOA) is a readiness for practice. Grid computing is a promising SOA to enable generalized resource sharing in a virtual organization across physical organizations. TCG enabled security with strong means for remote policy enforcement forms not only a practical and near-term realizable service oriented methodology for service providers, but also a sufficiently strong evidence for commercial enterprises including financial institutions to be convinced with high confidence to go for and offer out-sourced enterprise computing and grid computing services.

Important next steps of the work include to support the standard grid middleware GT4. This is a challenging task involving standardization effort across areas of the grid, TCG and OS virtualization.

Acknowledgments

Greg Astfalk reviewed an early draft of this article and provided insightful comments and suggestions. Zhidong Sheng, Jing Zhan, Liqiang Zhang, Lihua Song, Qi Sun, Miao Zhang, Lei Zhang (of Wuhan University), Chunrun Chen, Shuming Liu, Wenbo Yan, Chuanjiang Yi (of Huazhong University of Science and Technology), Cheng Chen, Rong Chen, Pengchen Liu and Fengzhe Zhang (of Fudan University) participated in the implementation work. Chris Hsiung, Hai Jin, Wei Liu, Martin Sadler, Binyu Zang and Huanguo Zhang provided organisational, coordinative and logistic support to the Project. The authors would also like to thank Allan Tomlinson and anonymous reviews for reviewing and editing this article for inclusion in Information Security Technical Report of Elsevier.

Appendix A. Trusted computing

In recent years, increased reliance on computer security and the unfortunate fact of lack of it, particularly in the open-architecture computing platforms, have motivated many efforts made by the computing industry. Among these is the development of *Trusted Computing (TC)*. In 1999 five companies – Compaq, HP, IBM, Intel and Microsoft – founded *Trusted Computing Platform Alliance (TCPA)*. The motivation of TCPA was to add trust to open-architecture computing platforms. In 2003 TCPA achieved a membership of 190 plus companies, when it was incorporated to *Trusted Computing Group (TCG)* (Trusted Computing Group, 2003; Pearson, 2003). TCG is a vendor-neutral and not-for-profit organization for defining, specifying and promoting industrial standards for the TC technology. The TCG work has so far been developed to contain sufficient innovations and become a standard methodology for adding trust and security to open computing platforms.

TCG's approach to adding trust is to integrate to a computer platform a hardware module called *Trusted Platform Module (TPM)*. TPM has a tamper-protection property. It is intended by TCG that TPM can play the role of an in-platform trusted third party agent to enforce a conformed behavior for software systems running on the platform. TPM must be trusted to function properly as it is designed. Trust in TPM's correct functionality is underpinned by a number of elements. As an industrial standard body, TCG considers that this notion of trust is materialized by the following elements:

- The tamper-protection assumption of the TPM that the behavior of any of its inner component cannot be subverted by any external principal.
- Open specifications of the design for the hardware, software, firmware components, algorithms and protocols, which are used by the TCG. The open specifications facilitate expert review for minimizing the possibility of design errors.
- Standard processes and criteria for evaluation and certification of the system. TCG stipulates that evidence of engineering practice and industry review follow the Common Criteria (CC) certification results.
- Good engineering practices by the manufacturer, with standard approach to defining, guiding and industry review of manufacturing processes.

We believe that the above mechanism of trust is reasonable for establishing and maintaining a reliable behavior to be expected from a TPM. Although it was expressed in the guise of arguing the meaning of trust, that view is in fact a concern on whether the TCG technology may be misused, in particular by a few large companies, to stifle competitions. Given the fact that a TPM can indeed play the role of an in-platform agent with a conformed behavior which is designed out of control of the platform owner, the concern of technology misuse is an understandable one.

At any rate, we are fortunate to be able to avoid the issue of technology misuse. In this article when we speak of trust, we confine ourselves to the idea of a platform system having an expected behavior supporting grid, federated or collaborated computing applications in which principals accept that they should comply with a commonly agreed behavior. In our grid computing model, conformed behavior is a requirement rather than a problem, and hence misuse of trust is not an issue.

A.1. *TC working principle*

The following four notions are at the core of the TC technology.

A.1.1. *Trusted platform module (TPM) – an in-platform trusted third party*

This is a tamper-protection hardware module uniquely integrated to a platform. The tamper protection is in the following two senses:

Shielded locations: These are places (memory, register, etc.) which have the hardware base inside the TPM and may be extended to a place outside the TPM via the supporting software system. A shielded location holds information which cannot be accessed by any external principal in any way to violate data confidentiality. The information held in a shielded location can only be used in the TCG designed ways by ''protected capabilities'' (see below). Information protected by shielded locations includes cryptographic keys and some system integrity metrics which are held by a number of hardware registers inside the TPM.

Protected capabilities: These are designated computations and operations performed by the TPM which have exclusive permission to access shielded locations. Because of the need for accessing shielded locations which are not available to any principal external to the TPM, a protected capability cannot be controlled or subverted by any of such principals in any non-pre-designed manner. Protected capabilities include: cryptographic operations inside the TPM such as the generation of random numbers and cryptographic keys, encryption, decryption and digital signature generation using keys in shielded locations; TPM functional operations which are related to measuring, storing and reporting of system metrics (see below), and TPM's system operations which maintain power detection, counters, etc.

The tamper protection is an assumption. Under this assumption, any principal external to the TPM is considered a potential adversary, and this even includes the owner of the TPM. This assumption is an important concept in TCG. Using the tamper-protection assumption it is intended that a TPM can indeed play the role of an in-platform trusted third party to protect some important data and perform some designated computations and functions.

It is actually reasonable to believe the validity of the tamper-protection assumption. The validity rests on an inequality between the cost of making a hardware chip with a tamper-protection quality and that of subverting it. This is somewhat analogous to the following fact in cryptography: designing a hard problem using an NP witness element is easier than solving the problem without the witness.

The integration between a TPM and a platform is also tamper protected. We note that this property will be important in our application of TC to realize a conformed policy for a VO.

A.1.2. Root of trust for measurement (RTM)

The simplest form of platform measurement is the hardware configuration properties of the platform in which a TPM is integrated in a tamper-protection manner. A certificate of hardware configuration can be issued by a trusted third party to the TPM-platform *after* the integration of the TPM into the platform. The signing capability of the TPM can later report the platform's hardware configuration status to a remote querier (see ROR in Appendix A.1.4). In this simple form of RTM, the TTP is trusted (by the remote querier) that it will not issue the certificate of hardware configuration if the platform does not have the configuration specified in the certificate.

A TPM contains a plural number of registers inside the TPM called *Platform Configuration Register (PCR)*. A PCR is a 160-bit hardware register which must be realized in volatile storage. Upon either system startup or the system reset event, a PCR is always reset to the initial state with a default NULL value. The TPM is integrated in the platform's hardware system in such a manner: it can obtain (or ''eavesdrop'', or ''overhear'', or ''wiretap'', or whatever you like to call it) any binary executable code which is loaded from an external device to the input–output hub in the CPU via the input–output bus. The ''eavesdropped'' copy of the binary executable is hashed inside the TPM and stored in a PCR in the PCR extension formula (1). The hash total valued of a PCR value in that formulation is called an RTM measurement result, which is in fact a digest image of the software measurement in the platform system (another name: the RTM integrity metric). The hash total formula permits a PCR to cumulatively record the RTM integrity metric results in an unbounded fashion. The stored platform environment status is maintained until system reboot.

A more advanced form of platform measurement is on its software configuration properties. At the platform boot time, the TPM measures the system's data integrity status. The measurement starts from the integrity of BIOS, then that of OS and finally to applications. Note that the lowest part of the program code of the RTM is also called the *core RTM (CRTM)* which is a firmware implemented instruction code programmed to measure the very first piece of software system a platform will be running. With CRTM, it is in principle possible to establish a desired platform environment by loading only well behaved systems. Although a practical realization is so far still beyond commercial use. New thoughts on virtualization of operating systems have been proposed, e.g., OpenTC, to be a way round the problem.

A.1.3. Root of trust for storage (RTS)

The RTS is a computing mechanism which realizes TPM-enabled shielded locations in such a manner that they are able to hold information of a size not bounded to the (usually small) size of the TPM. There are two ways to achieve the RTS for two different services, respectively. One of the storage service is for storing information which requires confidentiality protection and the RTS locations for this use are usually outside the TPM in an external persistent storage (e.g., a hard disk drive). The other is for storing the RTM integrity metrics and the RTS locations for this use are usually inside the TPM.

The RTS location for storing confidentiality data is a straightforward application of the well-known cryptographic key management technique. At the TPM initialization time, the owner of the TPM creates a *storage root key (SRK)* in its shielded locations. The SRK a public/private key pair with the private key residing in a shielded location inside the TPM. Now consider a tree-structured hierarchy of key management system in which the SRK is in the root of the tree. The public key of the SRK is used to encrypt a plural number of children ''blobs.'' Each of the children blobs can be a ''wrapped key blob'' (encrypted key) or a ''wrapped data blob.'' A key in a wrapped key blob can be a symmetric key, a private key for decryption or signature generation uses. A symmetric key in a key blob can further encrypt a plural number of children blobs, and the same structure of the hierarchy is maintained downwards, …

The RTS location for storing the RTM integrity metrics of the system software is PCRs (see RTM in Appendix A.1.2).

Related to these two RTS services, in TCG there is a notion of ''migratable TPM information'' *(TPM migration)*. The TPM migration mechanism allows an authorized user of a TPM to securely transfer a secret in a shielded location of the TPM (the one the user is authorized to use) to a shielded location of another TPM of the user's choice. This mechanism permits a user to move her/his cryptographic credentials and/or mission-critical data from one trusted platform to another. This is a necessary security service for, e.g., a situation when the user changes platforms. While some user credentials and data secured in the RTS under the key management system of the SRK tree may be rendered migratable, information in the RTS for storing system integrity metrics is non-migratable.

A.1.4. Root of trust for reporting (RTR) and remote platform attestation

The TPM can report to a remote requester a RTM result regarding an executable running in the platform system. Here, the RTM result has been recorded in an RTS protected location. This service is achieved using a well-known cryptographic protocol technique of challenge-response: a remote querier challenges the TPM-platform with a random number, and the TPM-platform responses by the TPM signing the random challenge. The protocol enabling this TCG feature is called *remote platform attestation protocol*.

Remote platform attestation is a very innovative part in TCG. It is highly relevant to secure collaborated computing using the TCG technology. With remote platform attestation, a remote principal as a stakeholder in a collaborated computing job can be assured of a conformed behavior of a platform which is required by the secure collaborated computing application. Under the tamper-protection assumption of the TPM, the remote stakeholder knows that the conformed platform behavior cannot be easily subverted, not even by the platform owner.

REFERENCES

Adabala Sumalatha, Chadha Vineet, Chawla Puneet, Figueiredo Renato, Jose Fortes, Krsul Ivan, et al. From virtualized resources to virtual computing grids: the in-VIGO system. Future Gener Comput Syst April, 2005;21(6).

Bair R, editor. Agarwal D, et al. (contributors). National collaboratories horizons. Report of the August 10–12, 2004. In: National Collaboratories Program Meeting. The U.S. Department of Energy Office of Science.

Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, et al. Xen and the art of virtualization. In: Proceedings of the 19th ACM symposium on operating systems principles; 2003. p. 164–77.

Chen B, Morris R. Certifying program execution with secure processors. In: Proceedings of ninth hot topics in operating systems symposium; 2003. p. 133–8.

Chen L, Landfermann R, Loehr H, Rohe M, Sadeghi A-R, Stüble C. A protocol for property-based attestation. In: First ACM workshop on scalable trusted computing (STC'06); November 3, 2006. p. 7–16.

Common Criteria. Available from: <www.commoncriteriaportal.org>.

Dragovic B, Fraser K, Hand S, Harris T, Ho A, Pratt I, et al. Xen and the art of virtualization. In: Proceedings of the ACM symposium on operating systems principles, Bolton Landing, NY; October 2003. p. 164–77.

Efstathopoulos P, Krohn M, VanDeBogart S, Frey C, Ziegler D, Kohler E, et al. Labels and event processes in the asbestos operating system. In: Proceedings of the 20th ACM symposium on operating systems principles. ACM Press; 2005. p. 17–30.

Figueiredo RJ, Dinda PA, Fortes JAB. A case for grid computing on virtual machines. In: Proc. 23rd international conference on distributed computing systems (ICDCS'03), Providence, RI; 2003. p. 550.

Foster I, Kesselman C. The grid: blueprint for a new computing infrastructure. Morgan Kaufmann; 1999.

Foster I, Kesselman C, Tsudik G, Tuecke S. A security architecture for computational grids. In: 5th ACM conference on computer and communications security; 1998. p. 83–92.

Foster I, Kesselman C, Tuecke S. The anatomy of the grid: enabling scalable virtual organizations. Int High Perform Comput Appl 2001;15(3):200–22.

Garfinkel Tal, Pfaff Ben, Chow Jim, Rosenblum Mendel, Boneh Dan. Terra: a virtual machine-based platform for trusted computing. In: Proceedings of the nineteenth ACM symposium on operating systems principles. ACM Press; 2003. p. 193–206.

Globus Toolkit 4. <www-unix.globus.org/toolkit/>.

Haldar V, Chandra D, Franz M. Semantic remote attestation: a virtual machine directed approach to trusted computing. In: USENIX VM; 2004.

Härtig H, Hohmuth M, Feske N, Helmuth C, Lackorzynski A, Mehnert F, et al., The Nizza secure-system architecture. In: Proc. of collaborate com; 2005.

Jaeger Trent, Sailer Reiner, Zhang Ziaolan. Analyzing integrity protection in the SELinux example policy. In: Proceedings of the 12th USENIX security symposium. USENIX; August 2003. p. 69–75.

Jaegar T, Sailer R, Shankar U. PRIMA: policy-reduced integrity measurement architecture, SAC-MAT'06. In: Proceedings of the tenth ACM symposium on Access control models and technologies, C9. Lake Tahoe, CA, USA: ACM Press; June 7, 2006. p. 19–28.

Krsul Ivan, Ganguly Arijit, Zhang Jian, Fortes Jose AB, Figueiredo Renato JO. VMPlants: providing and managing virtual machine execution environments for grid computing. SC; 2004.

Lie D, Thekkath C, Mitchell M, Lincoln P, Boneh D, Mitchell J, et al. Architectural support for copy and tamper resistant software. In: Proceeding of international conference on architectural support for programming languages and operating systems; 2000. p. 168–77.

Lie D, Thekkath C, Horowitz M. Implementing an untrusted operating system on trusted hardware. In: Proceedings of the 19th ACM symposium on operating systems principles. ACM Press; 2003. p. 178–92.

Lorch M, Basney J, Kafura D. A hardware-secured credential repository for grid PKIs. In: Proc. of 4th IEEE/ACM international symposium on cluster computing and the grid. New York: IEEE Press; 2004. p. 640–7.

Loscocco P, Smalley S. Integrating flexible support for security policies into the Linux operating system. In: Proceedings of the FREENIX track: 2001 USENIX Annual Technical Conference table of contents. USENIX; 2001. p. 29–42.

Marchesini John, Smith Sean. SHEMP – secure hardware enhancement for MyProxy. Hanover, New Hampshire: Department of Computer Science, Dartmouth College; February 2005. Technical Report TR2005-532.

National Computer Security Center. Integrity in automated information systems September 1991.

NGSCB. Microsoft Corporation. Microsoft virtual server. Available from: <www.microsoft.com/windowsserversystem/virtualserver/default.mspx>; 2005.

Open Grid Forum. Overview of the GSI. <www.globus.org/security/overview.html/>.

OpenTC. Available from: <www.opentc.net/>.

Pearson Siani, editor. Trusted computing platforms. Prentice Hall; 2003.

Peinado M, Chen Y, England P, Manferdelli J. NGSCB: a trusted open system. In: Proc. of 9th Astralasian Conference on Information Security and Privacy. ACISP; 2004. p. 86–97.

Peinado M, Chen Y, England P, Manferdelli J. NGSCB: a trusted open system. Microsoft Corporation. Available from: <microsoft.com/yuqunc/papers/ngscb.pdf>; January 2005.

Rau S. The trusted computing platform emerging as industry's first comprehensive approach to IT security. IDC Executive Brief. Available from: <www.trustedcomputinggroup.org/news/Industry_Data/IDC_448_Web.pdf>; February 2006.

Sadeghi A-R, Stüble C. Property-based attestation for computing platforms: caring about properties, not mechanisms. In: New Security Paradigm Workshop (NSPW); 2004.

Sailer Reiner, Jaeger Trent, Zhang Xiaolan, van Doorn Leendert. Attestation-based policy enforcement for remote access. In: ACM conference on computer and communications security; 2004a. p. 308–17.

Sailer R, Zhang X, Jaeger T, van Doorn L. Design and implementation of a TCG-based integrity measurement architecture. In: Proceedings of the 13th USENIX security symposium; 2004b. p. 223–38.

Sailer Reiner, Valdez Enriquillo, Jaeger Trent, Perez Ronald, van Doorn Leendert. John Linwood Griffin and Stefan Berger. sHype: a secure hypervisor approach to trusted virtualized systems. IBM Res Rep 2005a.

Sailer R, Valdez E, Jaeger T, Perez R, van Doorn L, Griffin JL, et al. sHype: secure hypervisor approach to trusted virtualized systems. Technical Report RC23511, Yorktown Heights, NY: IBM TJ Watson Research Center; February 2005b.

Sandhu Ravi, Zhang Xinwen. Peer-to-peer access control architecture using trusted computing technology. In: Proceedings of the 10th ACM symposium on access control models and technologies (SACMAT), Stockholm; 2005. p. 147–58.

Shankar Umesh, Jaeger Trent, Sailer Reiner. Toward automated information-flow integrity for security-critical applications. In: Proceedings of the 13th annual network and distributed systems security symposium. Internet Society; 2006.

Shapiro J, Smith J, Farber D. EROS: a fast capability system. In: Proceedings of 17th ACM symposium on operating systems principles. ACM Press; 1999. p. 170–85.

Shi Weidong, Lee Hsien-Hsin S, Lu Chenghuai, Ghosh Mrinmoy. High speed memory centric protection on software execution using one-time-pad prediction. Atlanta, GA: Georgia Institute of Technology; July 2004. Report GIT-CERCS-04-27.

Shieh A, Williams D, Sirer E, Schneider F. Nexus: a new operating system for trustworthy computing. In: Proceedings of the 20th ACM symposium on operating systems principles, WIP Session; 2005. p. 1–9.

Suh G, Clarke D, Gassend B, van Dijk M, Devadas S. AEGIS: architecture for tamper-evident and tamper-resistant processing. In: Proceedings of the 17th annual international conference on supercomputing; 2003a. p. 160–71.

Suh GE, Clarke D, Gassend B, van Dijk M, Devadas S. Efficient memory integrity verification and encryption for secure processors. In: Proc. annual IEEE/ACM international symposium on microarchitecture (MICRO); 2003b. p. 339–50.

TrouSerS. The open-source TCG Software Stack. <trousers.sourceforge.net/>.

Trusted Computing Group. Available from: <www.trustedcomputinggroup.org>.

Trusted Computing Group. Trusted computing platform alliance (TCPA) main specification, version 1.1a. Republished as Trusted Computing Group (TCG) main specification, version 1.1b, <www.trustedcomputinggroup.org>; 2001.

Trusted Computing Group. TCG TPM specification 1.2. Available from: <www.trustedcomputinggroup.org>; 2003.

The VMware software package. Available from: <www.vmware.com>.

Server Utilization. <www.serverwatch.com/>.

Zeldovich N, Boyd-Wickizer S, Kohler E, Mazieres D. Making information flow explicit in HiStar. In: Proceedings of 7th USENIX symposium on operating systems design and implementation. USENIX; 2006. p. 279–92.