Trusted Virtual Domains – Design, Implementation and Lessons Learned

Luigi Catuogno[1], Alexandra Dmitrienko[1⋆] , Konrad Eriksson[2], Dirk Kuhlmann[3], Gianluca Ramunno[4], Ahmad-Reza Sadeghi[1], Steffen Schulz[1], Matthias Schunter[2], Marcel Winandy[1], Jing Zhan[1,5⋆⋆]

[1] Horst Görtz Institute for IT Security, Ruhr-University Bochum, Germany
{luigi.catuogno, alexandra.dmitrienko, ahmad.sadeghi, steffen.schulz, marcel.winandy, jing.zhan}@trust.rub.de
[2] IBM Research – Zurich, Switzerland
KON@zurich.ibm.com, mts@zurich.ibm.com
[3] Hewlett Packard Laboratories – Bristol, England
dirk.kuhlmann@hp.com
[4] Dip. di Automatica e Informatica, Politecnico di Torino, Italy
ramunno@polito.it
[5] Wuhan University, Department of Computer Science, Wuhan, China

Abstract. A Trusted Virtual Domain (TVD) is a coalition of virtual machines and resources (e.g., network, storage) that are distributed over multiple physical platforms and share a common security policy. The concept of TVDs and their usage scenarios have been studied extensively. However, details on certain implementation aspects have not been explored in depth yet, such as secure policy deployment and integration of heterogeneous virtualization and trusted computing technologies. In this paper, we present implementation aspects of the life cycle management of TVDs. We describe the components and protocols necessary to realize the TVD design on a cross-platform architecture and present our prototype implementation for the Xen and L4 microkernel platforms. In particular, we discuss the need for and the realization of intra-TVD access control, a hypervisor abstraction layer for simplified TVD management, necessary components of a TVD policy and revocation issues. We believe that these integration details are essential and helpful inputs for any large-scale real-world deployment of TVD.

Keywords: trusted virtual domain, security, virtualization, management

1 Introduction

A Trusted Virtual Domain (TVD) [1–4] is a coalition of virtual machines that trust each other based on a common security policy. The policy is uniformly

⋆ Supported by the Erasmus Mundus External Co-operation Window Programme of the European Union

⋆⋆ Affiliated with Ruhr-University Bochum and partly sponsored by the China Scholarship Council(CSC) at the time of writing this paper

enforced, independent of physical boundaries. TVDs build up on virtualization techniques to provide confinement boundaries for a protected execution environment that are typically distributed over several physical platforms. Different instances of several TVDs can co-exist on the same physical platform. Communication within the TVD (intra-TVD communication) is possible through the use of shared resources such as network interface and storage. The underlying virtual machine monitor (VMM) isolates different TVDs and enforces access control to TVD resources according to the underlying TVD policy.

TVDs are different from traditional access control models in that they are more abstract and platform independent. This allows consistent enforcement of a security policy regardless of individual implementations and physical infrastructure topology. While conceptually easy, the implementation of TVDs requires integration and automation of sophisticated security mechanisms like secure network virtualization [5], secure hypervisors [6], trusted channels [7–9] and virtualized hardware security modules [10–13]. Further, scalability and seamless integration of different platforms are essential features of such an infrastructure.

The conceptual simplicity of TVDs also suggests that they may succeed where previous attempts on access control in multi-domain environments have been ineffective or caused considerable operational overhead. We consider them particularly well-suited for large distributed environments like enterprise networks, cloud computing, personal area networks or e-health infrastructures.

As a result, a number of research projects pursue the developments of TVD frameworks. In particular, we mention the European Multilaterally Secure Computing Base (EMSCB) [14] and the Open Trusted Computing (OpenTC) [15, 16] projects.

Unfortunately, despite large research effort, there are few detailed descriptions of full-featured TVD implementations. We believe this is due to the high internal complexity of such an infrastructure and because the required effort to integrate the highly sophisticated subsystems is easily underestimated. Indeed, in the development of our prototype we discovered a variety of unanticipated issues that are not addressed in existing works, e.g., the issue of different privileges inside a TVD or the problem of revocation.

Contribution and outline. In this paper we present the design and implementation details of our TVD architecture. We describe the components and protocols needed for life-cycle management of TVDs (Section 3). In particular, we show how to integrate the trusted computing functionality to securely distribute the TVD policy to local policy enforcement components. We describe our implementation of a virtual data center use case (Section 4), which includes: (i) the realization of intra-TVD access control, (ii) a hypervisor abstraction layer for simplified TVD management, (iii) the definition and usage of a TVD policy, and (iv) revocation of TVD components. Moreover, we discuss practical aspects which we encountered as lessons learned (Section 5) during the development of our implementation. Finally, Section 6 elaborates on related work.

2 Design Goals

In this section we consider the main security objectives of TVDs, define our assumptions and threat model, and discuss the security requirements. Our implementation of TVD infrastructure address the following main security objectives:

1. *Secure TVD membership and revocation*: Virtual and/or physical machines that join or leave the TVD should fulfill the requirements of a well defined policy. This policy may change during the lifetime of the TVD, for instance, revoked machines must be excluded from the TVD.
2. *Secure TVD communication*: All members of a TVD, and shared resources over the TVD, are connected through a virtual network that can span over different platforms, and that is strictly isolated. Non-members are not allowed to access such a network.
3. *Intra-TVD security*: Some members of a TVD may have higher privileges than other members. Hence, the communication within the TVD needs to be further restricted by access control enforced by certain TVD members with specific responsibilities, e.g., for TVD management.

In this paper, we do not address *inter-TVD* communication, although specific applications may require some communication between members of different TVDs according to any *inter-TVD* information flow policy.

2.1 Assumptions, Threat Model, and Requirements Analysis

For the correct function and security of the TVD, we assume that the TVD policy is consistent, i.e., it does not contain conflicting statements. Moreover, we assume that the trusted computing base (TCB), which enforces the TVD policy, works correctly, i.e., it provides the specified functional and security properties.

Our threat model considers adversaries who can access communication channels (e.g., by sniffing network traffic) and compromise components of the TVD. We assume the TCB on each platform cannot be compromised at runtime, but it can be replaced or altered between two bootstrapping phases (i.e., binaries can be replaced or modified). However, in this case the modification should be detected.

On the other hand, facing runtime compromise of the TCB is still an open problem and leads to significant extensions of this approach that is beyond the scope of this paper. Research work on runtime aspects can be found, for example, in [17–20].

Based on the adversary model above, there are threats against each security objective of the TVD: First, secure membership can be compromised by replacing VMs or components of the TCB that enforce the membership, e.g., to tamper with their file-image on the storage backend. Hence, a primary security requirement is to verify the integrity of TVD members, including VMs and the TCB. Additionally, if a TVD member (possibly because of being compromised) is revoked, the adversary could attack the policy update mechanism that informs other members about the revocation. Possible attacks are impersonating

the TVD component that is responsible for updating and distributing the policy. The adversary could send a forged policy or replay an old one. Hence, the TVD infrastructure requires authentication of any policy distribution or update.

Second, by eavesdropping or manipulating on communication channels, the adversary could gain information that should not be leaked. Hence, secure channels between TVD members are required to provide authenticity, confidentiality, and integrity.

Third, if adversaries control a TVD member, they can access other members and resources of this TVD via the common TVD communication infrastructure. But if the adversaries control only a member of low privileges, members with higher privileges should still be protected. Hence, we need to identify TVD members and limit access to TVD internal resources according to their privileges.

Finally and related to all threats mentioned above, the TCB of each involved platform has to provide isolation of execution environments — this is particularly important when VMs of different TVDs are executed on the same platform.

As mentioned before, our TVD implementation does not feature any mechanism to discover VMs that have been tampered at runtime. However, strict separation of execution environments generally allows confinement of misbehaving VMs, preventing the adversary to attack other VMs running on the same platform.

3 Design of TVD Infrastructure

In this section we define our general TVD architecture and introduce the most relevant components.

3.1 General TVD Architecture

Similar to existing approaches [5], our TVD architecture includes two main components. A central component TVD Master is complemented by TVD Proxies; one instance of such a proxy is running on each physical platform hosting a TVD and represents the local copy of TVD Master. TVD Master stores and enforces the corresponding policy TVD Policy for admission of physical platforms, whereas TVD Proxy enforces the policy for admission of VMs. TVD Master is a logical service and could be implemented either on a single server or in distributed way.

In our design, TVD Policy defines the security policy of the TVD in the following way. It includes:

1. Configurations of virtualization platforms that are trusted to run the TVD. We denote such platforms as TVD Platforms. TVD Platforms configuration is represented by integrity measurements[6] of a platform's trusted computing base (TCB).
2. Configurations of virtual machines (VMs) which are trusted to be executed on TVD Platforms and allowed to join the TVD. Such VMs are called TVD VMs. By VM configuration we mean integrity measurement of VM's binaries.

[6] Here calculated as a cryptographic hash values of the corresponding binaries

3. TVD Resources like network, storage or special devices that have to be available to individual TVD VMs or their groups.
4. Intra-TVD access rules describing access restrictions within the TVD.
5. Credentials necessary to establish secure communication channels, e.g., cryptographic keys or certificates of the TVD.

Each platform has one TVD Proxy for each TVD. Before a VM running on TVD Platform can join a TVD, the corresponding TVD Proxy has to be instantiated on the platform. During this instantiation, TVD Master deploys TVD Policy to local TVD Proxy. After deployment, TVD Proxy enforces the admission of VMs to the TVD locally on the respective platform. Figure 1 shows our TVD architecture (see also [21]).

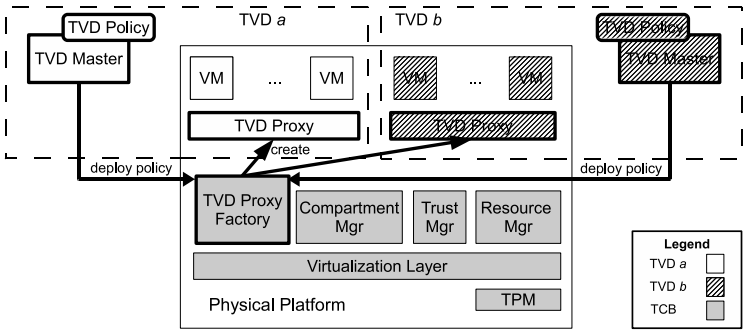

Fig. 1. General TVD architecture

To securely deploy and locally enforce TVD Policy, TVD Master has to rely on the TCB on each platform. The fundamental building block of the TCB is a virtualization layer that is able to run isolated VMs. The other main TCB components are TVD Proxy Factory, ResourceMgr, CompartmentMgr and TrustMgr.

TVD Proxy Factory is responsible for spawning new TVD Proxy instances. ResourceMgr provides access to basic virtualized resources TVD Resources like networking and storage. In case of TVD VMs, access to these resources is controlled by TVD Proxy. CompartmentMgr is responsible for managing virtual machines (compartments) and their attributes. It starts, stops, pauses VMs and attests their configuration, i.e., performs integrity measurement.

TrustMgr provides an abstraction of the underlying trusted computing functionality in hardware, here the Trusted Platform Module[7] (TPM) [22]. TrustMgr is responsible for generation of cryptographic keys (and corresponding certificates) that are protected by the TPM and are bound to the configuration (integrity measurements) of the components of the underlying TCB. Integrity measurements, keys, and certificates allow a remote party to establish a *trusted*

[7] Note that it could be other suitable security modules

channel to the platform, i.e., a secure channel (providing confidentiality, integrity and authenticity) that is bound to the integrity measurements of the endpoint(s) [8, 7, 9].

For the management of the TVD we present the following main protocols: TVDDeploy(), TVDJoin(), TVDLeave() and TVDUndeploy(). We will explain them in detail and briefly discuss the problem of policy updates and policy revocation.

3.2 TVD Deploy Protocol

The goal of the TVDDeploy() protocol is to deploy TVD Policy to local TVD Platform. The (remote) TVD Master attests to the trustworthiness of TVD Platform (compliance with the platform configuration defined in TVD Policy) and delivers the policy file to the trusted virtualization layer of that platform. More precisely, attestation in this context means validation of platform configuration of a remote party, e.g., integrity measurements that are stored in the TPM during the bootstrapping procedure. When receiving TVD Policy, TVD Proxy Factory creates TVD Proxy instance which is responsible for the local policy enforcement of that particular TVD.

The complete protocol TVDDeploy() is illustrated in Figure 2. It can be decomposed into two phases. The first one is the establishment of a trusted channel between TVD Proxy Factory of TVD Platform and TVD Master, the second one is creatating and configuring of TVD Proxy. The details of trusted channel establishment have already been presented in [8], which we adopted and extended for our purpose. The protocol is always initiated by TVD Proxy Factory of a local platform TVD Platform.

1. First, TVD Proxy Factory requests TVD Master for deployment where $nonceA$ and $nonceB$ denote the corresponding nonces for freshness. Moreover, we denote the signing and verification key of TVD Master with TVDMasterPKsignand TVDMasterSKsignrepsectively.
2. Next, TVD Master attests the requesting platform by verifying the binding certificate $certBind$. In our implementation, $certBind$[8] is issued by TrustMgr using the TPM. In particular, $certBind$ includes the quantities $PKbind$, $PKsign$ and $TPMdataSig$ denoting the public part of the binding key pair ($PKbind$, $SKbind$), the public part of the local platform's signing key ($PKsign$, $SKsign$), and the signature under $SKsign$ on $PKbind$, the configuration of TVD Proxy Factory m and on $nonceB$. The key pairs ($PKbind$, $SKbind$) and ($PKsign$, $SKsign$) are generated by the TPM and their secret parts can only be used by the TPM. Moreover, the access to $SKbind$ is restricted to the platform configuration $TCBconf$. $SKbind$ and $SKsign$ are stored outside the TPM only in encrypted form.[9] We denote the corresponding ciphertexts with $ESKbind$ and $ESKsign$ respectively.

[8] Note that $certBind$ is not an X.509 certificate

[9] In our implementation they are encrypted under the TPM's storage root key (SRK) of the TPM.

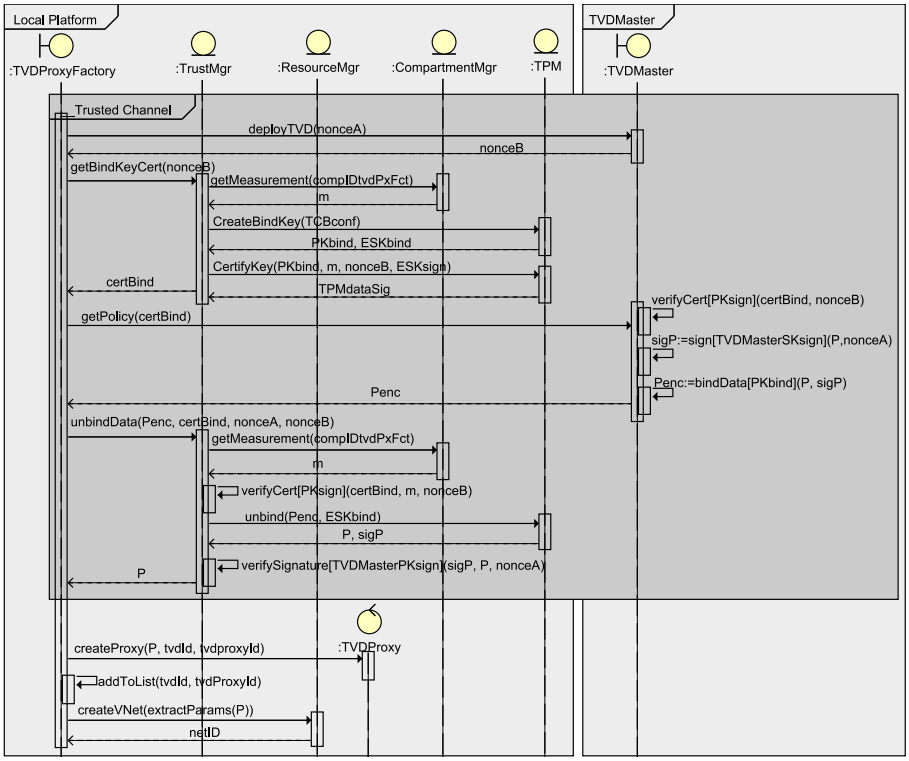

Fig. 2. TVD Deploy Protocol

3. After verifying *certBind*, TVD Master checks whether the platform configuration $TCBconf$ complies with its trust policy and if positive, binds TVD Policy to this certificate. For this, it first signs TVD Policy denoted as P together with the nonce *nonceA*. using its signing key (TVDMasterPKsign, TVDMasterSKsign). Then, TVD Master encrypts the TVD Policy P and the signature $sigP$ with $PKbind$. The result $Penc$ is sent to the local platform.
4. TrustMgr can only unbind (decrypt) $Penc$, if the current platform configuration is $TCBconf$, and the corresponding verifications are successful. In particular, TrustMgr verifies (i) $sigP$ whether TVD Master is authorized to define this TVD on this TVD Platform and (ii) whether TVD Proxy Factory is the owner of the certificate *certBind* and, hence, it is allowed to use the key $SKbind$. This is done by checking the configuration (integrity measurement) m of TVD Proxy Factory, that should match the value from the certificate *certBind*.
5. TVD Proxy Factory then starts local TVD Proxy for this TVD and configures it according to the received TVD Policy. It also passes the corresponding

parameters to ResourceMgr to configure TVD Resources as defined in TVD Policy.

6. Finally the newly created TVD Proxy is added to the list of the TVDs already deployed to the local platform.

3.3 TVD Join Protocol

After TVD Policy is deployed to the local platform, VMs can join this TVD. To connect a VM to a TVD, the TVDJoin() protocol is executed as shown in Figure 3.

In our implementation TVD Proxy Factory maintains a list of TVDs deployed to the local platform. Hence, a VM requests a proxy identifier $tvdProxyID$ of the required TVD from TVD Proxy Factory. If this TVD has not been deployed yet to the local platform, TVD Proxy Factory first runs the TVDDeploy() protocol (see Section 3.2) to create TVD Proxy of the corresponding TVD. On success, VM gets the required identifier $tvdProxyID$ and is then able to communicate with TVD Proxy directly. Its join request $VMJoin$ gets accepted by TVD Proxy only if VM's integrity measurement m complies to the TVD Policy P. In this case, TVD Proxy asks ResourceMgr to connect the VM to the TVD and sends a message to CompartmentMgr to mark it with a label (e.g., red or green) corresponding to the TVD it was joined to.

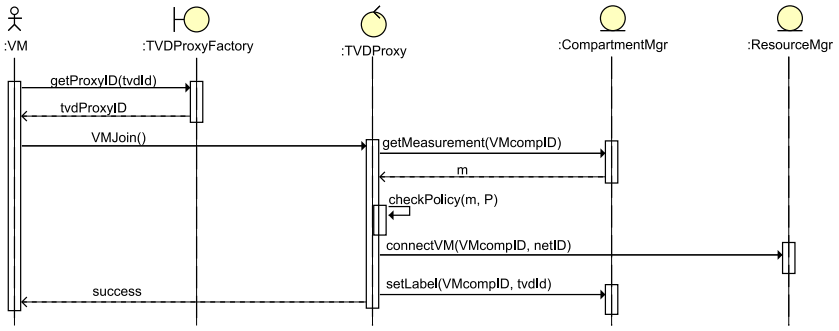

Fig. 3. TVD Join Protocol

3.4 TVD Leave and Undeploy Protocols

The TVDLeave() protocol is complementary to TVDJoin() and is depicted at the top of Figure 4: It involves the same components as TVDJoin() and is similarly initiated by VMs.

After TVDLeave() is completed, the TVD Proxy checks if there are any other VMs connected to the TVD. If not, it runs the TVDUndeploy() protocol in order

to remove itself from the local platform. TVDUndeploy() is shown at the bottom part of Figure 4. It may run after some delay (10 minutes in our case) only if no VM has initiated the TVDJoin() protocol before the timeout occurs.

3.5 Membership Revocation and Policy Update

The normal operation of a TVD requires mechanisms for membership revocation and policy updates as part of the general life cycle management. For instance, changes in resource assignment and access privileges[10] require the modification of the currently active TVD Policy, as well as the revocation of any TVD components instantiated based on the old policy.

In these cases, the TVD Master must revoke the old TVD Policy and distribute the new one to all hosts where the respective TVD is deployed. Care must be taken that all hosts are notified and hosts which are off-line or otherwise ignore the update are isolated from the updated TVD. To enforce this isolation, the low-level labels for access control to TVD Resources must be ephemeral and hard to guess (more details are given in Section 5.2). For resources like TVD Storage, which should support offline usage [23], we additionally propose to use lazy revocation [24, 25].

While a comprehensive revocation and isolation framework is not currently implemented, we present a simple extension to the TVDDeploy() protocol in Section 4.6 to show that such a framework can easily be added.

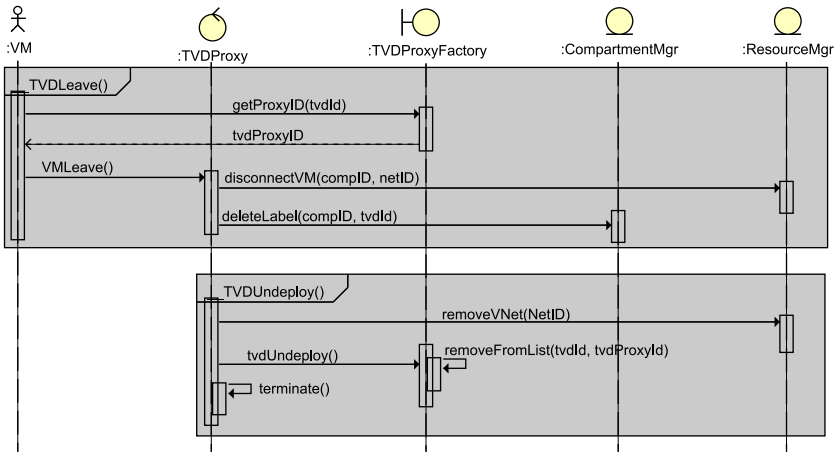

Fig. 4. TVD Leave and TVD Undeploy Protocols

[10] e.g., one or more VMs are no longer assigned to the TVD, or a network is no longer accessible by a certain TVD VM

4 Implementation

4.1 An Application Scenario

Our goal is to show advantages of the TVD concept in a scenario where the owner of a physical data center offers the operation of independent virtual data centers (VDCs) to several customers.

Customers can rent resources for a VDC from the data center operator. Within the resource constraints of their VDC, they can setup one or more TVDs by defining TVD Policy for each one. Customers can provide their policy definition and manage the TVD through TVD management consoles, which are also part of the corresponding TVD and run on either dedicated VDC management platforms or remotely, e.g., on the customer's laptop.

4.2 The VDC Demonstrator

Our VDC Demonstrator uses Trusted Computing technology to securely deploy a customer's data center as a fully virtualized system inside a physical data center. The main goal is to give more control to customers.

The demo setup is depicted in Figure 5. It consists of three Computing Platforms and one Management Platform. Two Computing Platforms are located in the data center and another one is connected to data center remotely. We use two switches to represent two different physical networks: The local network of the data center and the Internet. Inside the data center, the Management Platform (#3) is an accumulation of all servers required for normal operation of the data center, e.g., providing services for booting over network or assigning network addresses. Moreover, this platform realizes basic TVD infrastructure services like the TVD Master. It also provides the uplink to the Internet for the physical VDC as well as possibly defined TVDs. The Computing Platforms execute the actual workload, i.e., the TVD VMs. All machines in the data center are Xen based Linux hosts [26], while the remote Computing Platform (#4) is implemented on the L4/Fiasco microkernel [27] and represents a standard home PC or laptop.

The VDC Demonstrator runs two TVDs labeled as *red.tvd.net* and *blue.tvd.net*, or *red* and *blue* for short. Each TVD is comprised of a set of VMs and logical networks that are specified in the TVD Policy.

The remote platform (#4) is intended to be the remote administration console for the TVDs to which it is connected (*blue* and *red* in our demo). For each TVD, there is a dedicated management VM running isolated from other VMs on this platform. Depending on TVD Policy of each TVD, the management VM allows the local user of this platform to remotely access other VMs inside the TVD. We provide a graphical interface and allow the user to manage and access only those VMs that belong to the corresponding TVD and that the user has access to. The underlying network tunneling and policy enforcement is fully transparent to the user, who just has to define the (virtual) networks of his TVD and can then start VMs and join them to the TVD.

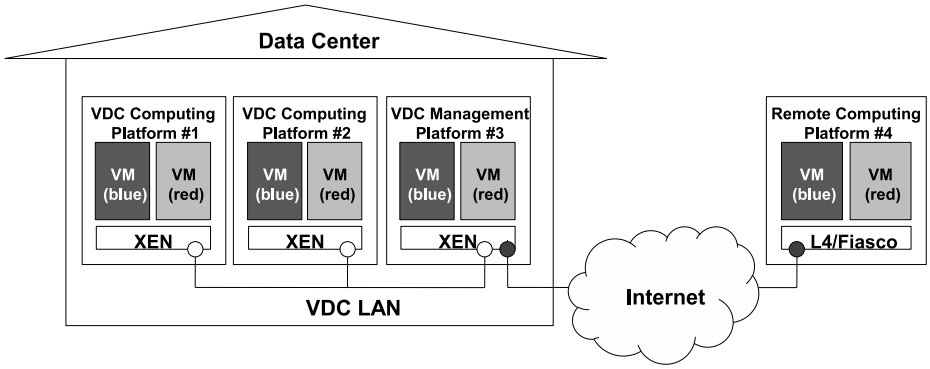

Fig. 5. The VDC Demonstrator Setup

4.3 Networked Hypervisor Abstraction

For automated remote management of compartments, we use the libvirt virtualization API [28]. Libvirt is a toolkit that features a common interface to use the virtualization functionalities of several mainstream open source virtual machine monitors including Xen [29], KVM [30], QEMU [31], and VirtualBox [32].

We implemented a libvirt interface to the L4 microkernel to allow libvirt to handle L4 in the same way as other supported hypervisors. As a result, we can integrate the L4 systems transparently into the management interface of the VDC and TVD administrators. Furthermore, to meet the security requirements of our project, we extend the libvirt with a Role Based Access Control (RBAC) module.

The RBAC module enforces the isolation in the TVD management by defining a distinguished role for the administrator of each TVD and by creating a separated view of the VDC resources for each role on a per-TVD basis. These views are defined through a set of rules that filter the access to the different resources according to their "ownership tag" that is the identifier of the TVD they belong to. The ownership tag is initially assigned to the administrator (i.e., it is associated to the corresponding role), and it is propagated to any VM the administrator requests to create and to join to the corresponding TVD.

For the integration in the Xen hypervisor, we have implemented a relay module that operates after the RBAC module. It intercepts requests on resources that are owned by the TVD, and lets CompartmentMgr on Xen carry out the associated security tasks, such as attestation and connection of the protected TVD Resources (e.g., encrypted disk storage).

4.4 Virtual Networking for TVDs

In context of VDCs, one usually assumes that the internal VDC network is trusted, while the communication channels to and from the VDC can be at-

tacked. Hence, we use virtual LAN (VLAN) [33] inside the VDC and labeled IPsec [34] in other cases.

VLAN-based virtualization provides easy and highly efficient integration into existing Ethernet networks, but it does not provide encryption and cannot defend against eavesdropping. IPsec-based network virtualization on the other hand is less efficient and more complex in comparison, but does not require a trusted physical network infrastructure and provides much more flexibility by running on top of IP, which is supported virtually everywhere.

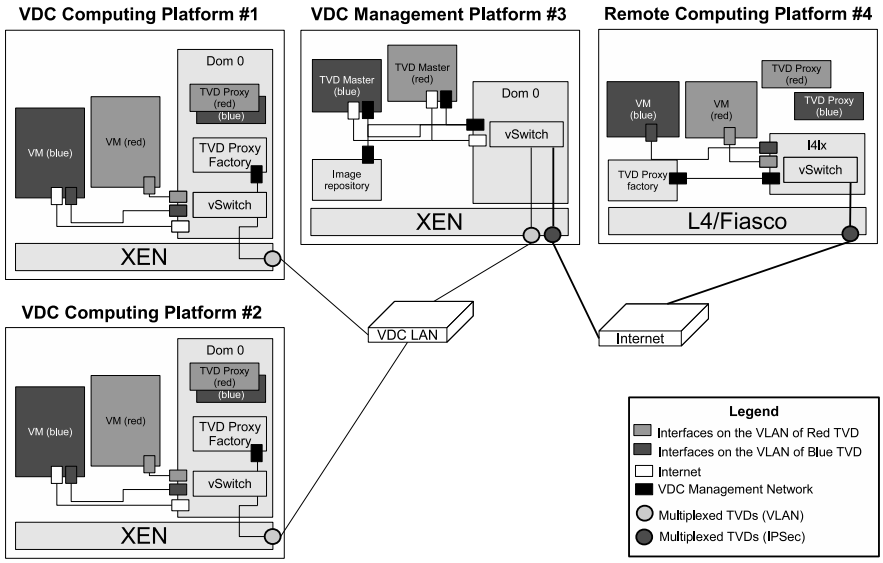

Fig. 6. Realization of virtual networks in the VDC demonstrator

We achieve an optimal trade-off between isolation of TVD VMs and remote management access to the TVD by introducing a separate management network for each TVD (see Figure 6). The main purpose of this network is to provide limited access to the hypervisor interface on each Computing Platform. This interface allows TVD owners (e.g., VDC customers) to create and manage the virtual machines (TVD VMs) and associated TVD Resources. To remotely manage the TVD, the TVD owner downloads a management VM provided by the TVD infrastructure provider and executes the TVDDeploy() and TVDJoin() protocols to join this VM to the TVD. According to TVD Policy, the management VM is joined to the respective networks, in this case the management network. This will enable the VM to access the hypervisor interface of all Computing Platforms that the TVD has expanded to.

Moreover, our virtual networks can also be used to provide access to other networks or TVDs to realize inter-TVD communication. For general Internet

access, this was implemented using a virtual bridge on the Internet gateway of the local physical TVD infrastructure. A corresponding virtual TVD network is provided for each TVD and connected to the individual TVD VMs as specified by the TVD Policy. While inter-TVD communication is possible this way, the resulting exchange point is common to all TVDs and the inter-TVD communication is not isolated from other TVDs. However, as noted earlier, actual inter-TVD communication is out of scope of this paper.

4.5 TVD Policy

TVD Policy is the machine-readable security policy of a TVD that specifies all components that can be admitted to a TVD. It contains a basic form of access control rules and specifies the configuration of the TVD resources. TVD Policy used in the VDC Demonstrator is an XML structure composed of two main parts: `tvd_nodes` and `tvd_layout` (see Appendix A for an example). The first one never leaves the TVD Master and specifies the identity of the systems (Computing Platforms) that can host TVD VMs. The TCB of such a system is remotely attested by the TVD Master during the first phase of the TVDDeploy() protocol (Section 3.2): If it matches one of the systems included in `tvd_nodes`, then the trusted channel is established and the second phase of the protocol can take place.

The systems identities are specified as collections (`systems`) of references to `reports`, each one generated by an `agent`. In this context, reports are, e.g., binary measurements, and agents are TPMs or measuring components of the TCB, like the L4 CompartmentMgr. The reports therefore represent building blocks for a whitelist of allowed systems. The `identity` clause for each agent defines how these reports shall be authenticated by the remote attestor. For example, in the case of TPM it could be the fingerprint of the public part of the attestation identity key, AIK, (or its X.509 certificate) used for the attestation.

The second part of the TVD Policy, `tvd_layout`, is sent to Computing Platforms via trusted channel during the second phase of the TVDDeploy() protocol. It is handed out to the TVD Proxy Factory that will spawn and configure the TVD Proxy with the policy settings. The latter are then used by the ResourceMgr to set up the TVD networks and will be used by the TVD Proxy later, during the TVDJoin() protocol, to check whether a VM can be admitted to the TVD or not. Allowed TVD VMs are expressed as collections (`systems`) of references to `reports` (as for `tvd_nodes`) to be matched and to `resources` to be attached to the VM, like logical networks and storage volumes. The configuration of each `resource` is also specified: the usual IP parameters and the encapsulation type (VLAN tagging, IPsec, etc.) for networks, the volume parameters and security features to apply (encryption, sealing) for storage. Other virtual resources with their configuration can be specified: They can also be indirectly attached to VMs, like virtual switches or VPNs setups.

The defined TVD Policy format allows the complete definition of TVD components and resources and can be further extended, e.g., to specify new types of resources. The structure of collections of reports simplifies the description of

the allowed systems and makes it more compact. However, if the list of systems is large, evaluating a system against the policy during a remote attestation may require a considerable amount of time; this aspect requires further tests and analysis. Finally, parsing XML requires the usage of large libraries unsuitable for minimalized components implementing, e.g., the trusted channel. In this case translating the XML format into a text format simpler to parse is required.

4.6 Revocation of TVD Components

Our prototype does not yet include a comprehensive revocation mechanism as motivated in Section 3.5. However, the previously described protocols and interfaces can be used to implement a rudimentary revocation system which we briefly describe here.

To revoke the authorization of a TVD Master to deploy a TVD in a TVD infrastructure, the certificate of the signing key used for authentication of TVD Policy must be revoked for the respective TVD infrastructure. Since an ordinary X.509 PKI is used to validate this signature, numerous revocation standards are available to propagate such information in the TVD infrastructure. Note, however, that it is up to the administrator of the physical infrastructure to revoke this authorization and that the TVD infrastructure used by a TVD can be composed of multiple administrative zones. Imagine for example a TVD with TVD Master M that should be migrated from data center A to B. After purchase of resources at B, the client (TVD owner) tells B to authorize M to deploy the TVD components. The TVD is now hosted by A and B simultaneously and the TVD owner can cancel the contract with A, so that any deployed resources in A are migrated to other parties, such as B. Then, A revokes the authorization of M to deploy new TVD components in A.

To revoke Computing Platforms or components of a TVD means to update TVD Policy, which lists all allowed Computing Platforms and TVD components available in a TVD, and to distribute this update to all Computing Platforms where the TVD is currently deployed to. A simple extension to the TVDDeploy() protocol (see Section 3.2) can be defined to implement this. It consists of a single message, *update*, that must be signed by the TVD Master under inclusion of the *nonceA* received in the previous TVDDeploy() protocol execution with the respective Computing Platform. TVD Proxy Factory, after successful verification of this message, will re-initiate the trusted channel establishment (which is part of TVDDeploy() protocol as described in Section 3.2). On success, it informs TVD Proxy about the updated TVD Policy and TVD Master can mark the host as updated. Since the policy transmitted by TVD Master is always signed, the additional signature and nonce are required here only to ensure that the *update* message cannot be replayed, which would potentially result in a denial of service on the Computing Platforms since the TVDDeploy() protocol is quite computation intensive.

Since such a TVD Policy update implicitly invalidates any previously deployed policy for that same TVD and host and the TVD Policy specifies all properties

and components of a TVD, this protocol can be used to enforce any required revocation and policy update in the TVD. TVD components that are not included in the updated TVD Policy anymore must be disconnected from the TVD. Components like TVD Storage can be isolated from the running TVD for manual inspection by the TVD owner, or lazy revocation schemes can be employed as noted in Section 3.5. Revoked Computing Platforms are also securely isolated: Since the TVDDeploy() protocol enforces remote attestation of the Computing Platform and its TCB, the deployment of the updated policy will fail for Computing Platforms that are not listed in TVD Policy anymore. As a result, these platforms cannot take part when the new low-level labels for TVD Resources are negotiated and thus get isolated from other TVD components.

A more scalable protocol is possible by leveraging the keys previously established in the trusted channel. From the description above it is clear, however, that our TVD design allows automated and highly flexible revocation and update of components. Naturally, more graceful approaches might be preferred if the revocation is not due to security compromise.

5 Lessons Learned

During the development of our TVD infrastructure we discovered some subtle issues which led to the experience we describe in this section. Some issues required changes in the design. As a result, we had to distinguish different network types within a TVD to separate normal operation and management. Other issues just complicated the development, but as a consequence raise the need for different implementation strategies in the future.

In the following, we motivate multiple logical networks within one TVD, discuss revocation issues, explain the need to negotiate labels by the TVD infrastructure, and, at the end, point out the need for a common hypervisor abstraction layer.

5.1 Multiple Logical Networks for Intra-TVD Networking

It became clear when designing our prototype that a TVD must support multiple logical networks with different sets of TVD VMs to achieve maximum isolation and yet allow certain privileged VMs to access management interfaces or external networks. Furthermore, customers will expect general Internet connectivity for some of their VMs as well as the ability to isolate some sets of TVD VMs from others. For example, a large server infrastructure will typically consist of multiple components like database backends, application layer proxies and systems for replication and hot failover. Such infrastructures use access control between components to enhance security and prevent unintended interactions. In real data centers, such access control is typically provided through extensive use of VLAN [33] and packet filtering in the network switches. However, such issues have not been discussed in context of TVD infrastructures before. Prior work [3] suggests to employ the TVD concept on multiple layers to control information

flow within a TVD. However, even a multi-layer TVD concept provides much less fine-grained access control than a simple network packet filter. For cloud computing services on the other hand, existing implementations like Amazon's Compute Cloud already support fine-grained access control and out-of-band security management of the system [35].

5.2 Revocation Issues

We described how revocation of platforms, TVD components and authorization of a TVD Master can be automated in a useful manner based on our TVD design. Although the idea is simple in case of TVDs, details like scalability and integration of graceful update and migration mechanisms remain to be solved. It also became apparent that revocation requires secure labeling of resources to enforce isolation. This appears to prohibit the use of simple label-based approaches such as VLAN [33], as it has been proposed in various previous works (e.g., [5]). If simple label-based virtualization is used, a compromised and revoked virtualization platform might be able to exhaustively search the only 2^{12} possible VLAN labels or otherwise derive the current label to produce a collision of resource labels, with the result that communication between the TVD and a revoked Computing Platform can be established again and secure isolation is breached.

As pointed out in Section 3.5, automated revocation is an integral part of the life cycle management in TVDs. We presented a basic implementation, however, more comprehensive and flexible solutions are necessary for large, automated infrastructures targeted by the TVD concept.

5.3 Labeling Scheme Needed to Control Access to Resources

Several issues must be considered for labeled shared resources. In Section 3.5 we argue that low-level labels should be ephemeral to allow effective exclusion of revoked parties. Another aspect are accidental label collisions between TVDs, which are well conceivable when a TVD is deployed to several TVD infrastructure providers at once.

We therefore propose low-level labels, i.e., labels that are negotiated on demand and used by the TVD infrastructure to enforce access control between TVD Resources and TVD VMs. If we consider labeled IPSec as a mechanism to implement a TVD Network, this becomes immediately obvious: To secure access to this resource and assure that revoked hosts are reliably excluded, a negotiation is needed between the corresponding Computing Platforms. In this case, the TVD Master will issue a new set of authorization tokens on each policy update to assure that Computing Platforms with outdated or revoked policies cannot participate in these negotiations. For the implementation of the label negotiation, the reader is referred to publications on group key exchange protocols such as [36].

5.4 Hypervisor Abstraction Layer Needs More Abstraction

In our work with different hypervisor and microkernel technologies, it became obvious that hypervisor abstraction is an important issue. In fact, the TVD concept itself is such an abstraction layer that specifies a set of resources and services which are provided independent from the underlying technology. We therefore used the libvirt hypervisor abstraction API because it allows for utilizing lightweight management tools and provides interfaces based on the Common Information Model (CIM) [37] for integrating with high level management suites. However, it turned out that the libvirt abstraction layer alone is not sufficient to cover some important aspects.

Inter Process Communication Services and VMs need to communicate with other services on a hypervisor platform, e.g., to execute the join procedure. This is performed via inter process communication (IPC). However, the libvirt abstraction does not cover IPC aspects. But to ease development of services and applications on a cross-platform level, a common IPC interface becomes necessary. Recent developments suggest the use of XML Remote Procedure Calls (XML-RPC [38]). But in our view it is less error prone, more efficient and light weight to define interfaces in a more abstract language, such as IDL [39], and let specialized compilers generate optimized code for the target architecture.

Automation of TPM Initialization TPMs were originally designed under the assumption that a human individual would perform a number of initial configuration actions. Consequently, several commands for currently available TPMs and corresponding BIOS setups were designed such as to ensure the physical presence of an operator during these steps. In data center scenarios, physical presence of an operator is an unreasonable assumption, and instead the support of remote initiation of TPM commands is required. However, during our integration efforts we had to conclude that it is not possible to fully automate the initialization process with the current generation of TPMs used in our setup (Infineon TPM v1.2). In the near future, deployment of such platforms will therefore rely on a certain amount of manual preconfiguraton with regard to enabling the TPM assuming ownership of it. This problem has since been reflected in the latest Provisioning Specifications of the TCG, and future generations of trusted platforms will allow to delegate these steps.

TPM Management The abstraction of different platforms in a TVD demands suitable management interfaces for remotely querying and configuring TPMs. This concern was mainly ignored during the TPM specification process by the TCG because they assumed interactions between human users and the TPM by means of software that already resides on the physical platform. Moreover, managing the hardware TPMs on physical platforms is a critical operation that may affect all TVDs that run instances on the machine. While it is important to guard the access to the corresponding functionality, our experiments have

shown that existing CIM agents lack support of fine grained access control that is required here. They often assume an all-powerful super user instead of different roles for, e.g., infrastructure operator and domain operator. In addition, the libvirt API is not well suited to host functionality that regards low-level aspects of the Trusted Computing hardware.

We therefore had to provide a dedicated API, although this was against our aim of supplying a unified management API. We developed a CIM model and a corresponding implementation that supports basic functionality such as reading out the endorsement key and important operational parameters, activating and deactivating the TPM, and so forth. We modeled the TPM as a CIM security device and service and covered most of the TPM information and functionality required for configuring a hosting platform for TVD components. Not included at this stage are functions of advanced TPM key management since they turned out to cut across multiple CIM profiles. On the other hand, our current working model already reflects that TPM related services might not yet be available at early configuration or boot up stages.

After all, our experiences suggest that an API suitable for managing all aspects of TVDs and trusted platforms will require at least an additional layer of abstraction in addition to the libvirt based approach presented above. In particular, they highlight the need of a comprehensive model covering all aspects of a TVD life cycle. This goes beyond what could be achieved at the abstraction level of libvirt. While the abstraction of IPC for services and applications needs a different mechanism, we believe that integration of TPM management should be possible by using a CIM based approach, provided that existing tools are extended with authorization control mechanisms.

5.5 TVD Policy Definition

When defining the language syntax to express the TVD Policy, we realized that one relevant requirement to meet is the trade-off between expressive richness and compactness. The need for a rich semantics stems from the number of areas the language must cover: integrity of TVD elements, basic access control (admitting a VM to a TVD or not) and configuration of TVD resources. The need for compactness originates from the nature of the components that must parse and enforce the policy. Since they are part of the TCB, their size must be as minimal as possible. In some case we found that specialized languages (like XACML [40] for access control) are too verbose, thus to reach the trade-off of effectiveness, we chose to write our own XML-based language.

Another lesson from our work is the degree of abstraction needed to describe TVD infrastructures: our achievement is that a whole network of systems can be represented in a single policy and all detailed mechanisms are abstracted away. The differences among the system architectures are related to: hypervisors, components implementing the virtual resources for VMs and system integrity (measurement and reporting). In this respect, our language can represent chains of trust that can include measurements held within the TPM and those kept in

software TCB components allowing for different levels of aggregation. Multiple ways to authenticate the integrity reports are supported. Moreover, complete VMs that are allowed in the TVD are identified in the policy by reference, i.e., through the digests of their root file system images, kernel and configuration file. The latter is expressed using a language independent from the hypervisor type, i.e., the libvirt XML format.

6 Related Work

Trusted Virtual Domains (TVDs) were first proposed in [1, 2]. Various applications of TVDs have already been shown and discussed, for instance, applying the TVD concept for secure information sharing [3], enterprise rights management [41], or virtual data centers [4, 42, 43]. In [4] and [42] the authors discuss the management of TVDs in data centers and present a high-level architecture for such a TVD infrastructure. Our TVD architecture is inspired by their work, however, we support different hypervisor architectures in an integrative solution and discuss new issues such as access control inside TVDs.

To the best of our knowledge, previous works on TVD establishment [3, 2, 4, 42] do not discuss in detail how to integrate trusted computing functionality in TVD management. In contrast, we present a detailed description of the protocols, including the integration of trusted computing functionality, needed to realize a secure life-cycle management of TVDs.

The closest work to ours is probably described in [43]. The authors describe an implementation which is similar to ours, but using Xen hypervisor only. They also mention attestation for integrity verification of the loaded software components in VMs and TCB components. However, their description of using trusted computing functionality is rather high level, whereas we describe in detail the protocols between TVD Master and local TCB components (TVD Proxy Factory, TVD Proxy, etc.). Moreover, we not only use attestation (embedded in the trusted channel between TVD Master and the local platform), but also use TPM-based data binding to deploy the TVD policy to local platforms and protect the credentials associated with the TVD policy. Finally, our implementation is cross-platform and works for both Xen and L4 virtualization.

An enhancement of TVD is to incorporate storage protection. The work in [43] extends the data center approach with controlled access to networked storage. In contrast, the work in [23] considers mobile storage devices, such as USB memory sticks, to incorporate in the TVD model. In that approach, additional components are introduced in the TCB of local platforms to feature identification and transparent encryption of storage devices, whereas existing components are enhanced to realize a dynamic centralized key management service. These works are complementary to ours and could be easily integrated into our architecture.

Techniques to isolate and manage the virtual networks of different TVDs are discussed in [5]. Basically, they propose a trusted virtual switch on each platform that uses VLAN tagging for local and IPsec for remote connections to implement strong isolation of networks. In [34], the authors propose security labels for IPsec

Security Associations to implement Multi Level Security (MLS) across networks. Their work was integrated into the Linux kernel and allows MAC enforcement between networked applications based on a mutually trusted TCB. Solaris Zones [44], an OS virtualization technology by Sun, is the first commercial product we know of that features networking virtualization based on labeled IPsec [45]. Our implementation is inspired by some of these ideas and is based on labeled IPsec and VLAN.

7 Conclusion

Trusted Virtual Domain (TVD) is a promising concept for secure management of virtualization platforms. We have presented the design and implementation of a TVD infrastructure where we considered the use case of Virtual Data Centers. Our design imposes only little overhead when compared to virtualized environments that do not use the TVD management framework. The protocol overhead for IPsec-based network virtualization is approximately 98 Byte per Ethernet frame for the additional EtherIP, IPsec and IP encapsulations.

We have described the life cycle management and implementation of a TVD based on Trusted Computing functionality. We have discussed automated revocation within TVDs as an integral part of the life cycle management of the TVD components. We motivated the use of separate management facilities for each TVD as well as basic access control mechanisms for TVD resources. The use of a hypervisor abstraction layer allows secure remote management of VMs and TVD resources for customers and data center administrators. Our prototype uses a simple user interface for such administrative tasks, but more feature-rich and convenient user frontends have to be developed in future for practical use in large-scale real-world environments. In particular, we are currently investigating effective mechanisms for handling revoked platforms or VMs. Moreover, we are also considering the deployment of TVDs in a broader range of applications scenarios, such as e-health.

Acknowledgments

We like to thank Thomas Fischer and David Plaquin from HP Labs for their input and contributions. Further, we thank Alexander Böttcher and Carsten Weinhold from Technical University Dresden for their support in developing for the L4 system. We also thank Christian Stüble from Sirrix AG for his input in various discussions.

References

1. Griffin, J.L., Jaeger, T., Perez, R., Sailer, R., van Doorn, L., Cáceres, R.: Trusted Virtual Domains: Toward secure distributed services. In: Proceedings of the 1st IEEE Workshop on Hot Topics in System Dependability (HotDep'05). (2005)

2. Bussani, A., Griffin, J.L., Jansen, B., Julisch, K., Karjoth, G., Maruyama, H., Nakamura, M., Perez, R., Schunter, M., Tanner, A., Van Doorn, L., Van Herreweghen, E.A., Waidner, M., Yoshihama, S.: Trusted Virtual Domains: Secure foundations for business and IT services. Technical Report RC23792, IBM Research (2005)
3. Katsuno, Y., Kudo, M., Perez, P., Sailer, R.: Towards Multi-Layer Trusted Virtual Domains. In: The 2nd Workshop on Advances in Trusted Computing (WATC 2006 Fall), Tokyo, Japan, Japanese Ministry of Economy, Trade and Industry (METI) (2006)
4. Berger, S., Cáceres, R., Pendarakis, D., Sailer, R., Valdez, E., Perez, R., Schildhauer, W., Srinivasan, D.: TVDc: managing security in the trusted virtual datacenter. SIGOPS Oper. Syst. Rev. **42** (2008) 40–47
5. Cabuk, S., Dalton, C.I., Ramasamy, H., Schunter, M.: Towards automated provisioning of secure virtualized networks. In: CCS '07: Proceedings of the 14th ACM conference on Computer and communications security, New York, NY, USA, ACM (2007) 235–245
6. Sailer, R., Jaeger, T., Valdez, E., Perez, R., Berger, S., Griffin, J.L., van Doorn, L.: Building a MAC-based security architecture for the Xen open-source hypervisor. In: ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference, IEEE Computer Society (2005)
7. Goldman, K., Perez, R., Sailer, R.: Linking remote attestation to secure tunnel endpoints. In: STC '06: Proceedings of the First ACM Workshop on Scalable Trusted Computing. (2006) 21–24
8. Asokan, N., Ekberg, J.E., Sadeghi, A.R., Stüble, C., Wolf, M.: Enabling fairer digital rights management with trusted computing. In: Proceedings of the 10th Information Security Conference (ISC). Volume 4779 of Lecture Notes in Computer Science., Springer (2007) 53–70
9. Armknecht, F., Gasmi, Y., Sadeghi, A.R., Stewin, P., Unger, M., Ramunno, G., Vernizzi, D.: An efficient implementation of trusted channels based on OpenSSL. In: STC '08: Proceedings of the 3rd ACM workshop on Scalable trusted computing, New York, NY, USA, ACM (2008) 41–50
10. Berger, S., Caceres, R., Goldman, K.A., Perez, R., Sailer, R., van Doorn, L.: vTPM: Virtualizing the Trusted Platform Module. In: Proceedings of the 15th USENIX Security Symposium, USENIX (2006) 305–320
11. Scarlata, V., Rozas, C., Wiseman, M., Grawrock, D., Vishik, C.: TPM virtualization: Building a general framework. In Pohlmann, N., Reimer, H., eds.: Trusted Computing. Vieweg-Verlag (2007) 43–56
12. England, P., Loeser, J.: Para-virtualized TPM sharing. In: TRUST 2008. Volume 4968 of LNCS., Springer (2008) 119–132
13. Sadeghi, A.R., Stüble, C., Winandy, M.: Property-based TPM virtualization. In: Information Security, 11th International Conference, ISC 2008. Volume 5222 of Lecture Notes in Computer Science., Springer (2008) 1–16
14. EMSCB Project Consortium: The European Multilaterally Secure Computing Base (EMSCB) project. `http://www.emscb.org` (2004)
15. The OpenTC Project Consortium: The Open Trusted Computing (OpenTC) project. `http://www.opentc.net` (2005)
16. Kuhlmann, D., Landfermann, R., Ramasamy, H.V., Schunter, M., Ramunno, G., Vernizzi, D.: An open trusted computing architecture – secure virtual machines enabling user-defined policy enforcement. Technical Report RZ 3655 (#99675), IBM Research (2006)

17. Nick L. Petroni, J., Fraser, T., Molina, J., Arbaugh, W.A.: Copilot - a coprocessor-based kernel runtime integrity monitor. In: Proceedings of the 13th USENIX Security Symposium. (2004) 179–194
18. Loscocco, P.A., Wilson, P.W., Pendergrass, J.A., McDonell, C.D.: Linux kernel integrity measurement using contextual inspection. In: STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing, New York, NY, USA, ACM (2007) 21–29
19. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: Proceedings of the 2003 Network and Distributed System Symposium. (2003)
20. Payne, B.D., Carbone, M.D., Lee, W.: Secure and flexible monitoring of virtual machines. In: Proceedings of the 2007 Annual Computer Security Applications Conference (ACSAC 2007). (2007)
21. Löhr, H., Sadeghi, A.R., Vishik, C., Winandy, M.: Trusted privacy domains – challenges for trusted computing in privacy-protecting information sharing. In: Information Security Practice and Experience, 5th International Conference, ISPEC 2009. Volume 5451 of Lecture Notes in Computer Science., Springer (2009) 396–407
22. Trusted Computing Group: TPM Main Specification, Version 1.2 rev. 103. (2007)
23. Catuogno, L., Manulis, M., Löhr, H., Sadeghi, A.R., Winandy, M.: Transparent mobile storage protection in trusted virtual domains. In: 23rd Large Installation System Administration Conference (LISA'09), USENIX Association (2009)
24. Backes, M., Cachin, C., Oprea, A.: Lazy revocation in cryptographic file systems. In: 3rd International IEEE Security in Storage Workshop (SISW 2005), December 13, 2005, San Francisco, California, USA. (2005) 1–11
25. Backes, M., Cachin, C., Oprea, A.: Secure key-updating for lazy revocation. In: Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security. Volume 4189 of Lecture Notes in Computer Science., Springer (2006) 327–346
26. Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., Barham, P., Neugebauer, R.: Xen and the art of virtualization. In: Proceedings of the ACM Symposium on Operating Systems Principles. (2003) 164–177
27. Hohmuth, M.: The Fiasco kernel: Requirements definition. Technical report, Dresden University of Technology (1998)
28. Libvirt project: libvirt virtualization API. `http://libvirt.org` (2008)
29. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP. (2007)
30. Qumranet Inc.: Whitepaper: Kernel-based virtualiztion machine. `http://www.qumranet.com/files/white_papers/KVM_Whitepaper.pdf` (2006)
31. Bellard, F.: QEMU, open source processor emulator (2008)
32. Sun Microsystems: Virtualbox (2008)
33. IEEE Computer Society: 802.11Q: Virtual Bridged Local Area Networks. (2003)
34. Jaeger, T., Butler, K., King, D.H., Hallyn, S., Latten, J., Zhang, X.: Leveraging IPsec for mandatory access control across systems. In: Proceedings of the Second International Conference on Security and Privacy in Communication Networks. (2006)
35. Amazon.com, Inc.: Amazon web services: Overview of security processes. Whitepaper, `http://aws.amazon.com` (2008)
36. Manulis, M.: Security-Focused Survey on Group Key Exchange Protocols. Technical Report 2006/03, Horst-Görtz Institute, Network and Data Security Group (2006)

37. Distributed Management Task Force: Common Information Model (CIM) Standards. http://www.dmtf.org/standards/cim/ (2009)
38. Winer, D.: XML-RPC Specification. (1999)
39. Object Management Group: OMG IDL Syntax and Semantics. (2002)
40. Organization for the Advancement of Structured Information Standards (OASIS): eXtensible Access Control Markup Language (XACML) v2.0. http://www.oasis-open.org/specs/ (2005)
41. Gasmi, Y., Husseiki, R., Sadeghi, A.R., Stewin, P., Stüble, C., Unger, M., Winandy, M.: Flexible and secure enterprise rights management based on trusted virtual domains. In: STC '08: Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing, ACM (2008)
42. Cabuk, S., Dalton, C.I., Eriksson, K., Kuhlmann, D., Ramasamy, H.G.V., Ramunno, G., Sadeghi, A.R., Schunter, M., Stüble, C.: Towards automated security policy enforcement in multi-tenant virtual data centers. Journal of Computer Science, Special Issue on EU's ICT Security Research, IOS Press (2009)
43. Berger, S., Cáceres, R., Goldman, K., Pendarakis, D., Perez, R., Rao, J.R., Rom, E., Sailer, R., Schildhauer, W., Srinivasan, D., Tal, S., Valdez, E.: Security for the cloud infrastructure: Trusted virtual data center implementation. IBM Journal of Research and Development **53** (2009) 6:1–6:12
44. Faden, G.: Solaris Trusted Extensions: Architectural Overview. http://opensolaris.org/os/community/security/projects/tx/TrustedExtensionsArch.pdf (2006)
45. Schuba, C.: Security Advantages of Solaris Zones Software. http://blogs.sun.com/schuba/resource/papers/ZonesSecurity-BP-010809.pdf (2008)

A TVD Policy Example

The following is an example of a TVD policy defining a Webserver connected to public Internet and an internal network towards its MySQL DB backend. A management VM is defined by default and allows to manage the TVD through the special management network.

```
<?xml version="1.0" encoding="UTF-8"?>
<tvd_policy id="blue.tvd.opentc.net">
  <tvd_nodes>
    <attestation>
      <agent name="TPM_1.2_default">
        <verification type="PrivacyCA">
          <identity algo="x509_cert">[base64 x.509 cert]</identity>
        </verification>
        <verification type="AIK">
          <identity algo="sha1_pubkey">[fingerprint of AIK public part]</identity>
        </verification>
      </agent>
    </attestation>
    <authentication>
      <agent name="VDC_nodes">
        <verification type="CA">
          <identity algo="x509_cert">[VDC x509 CA certificate]</identity>
        </verification>
      </agent>
    </authentication>
    <reports>
      <report type="tpm_pcrs" name="HP_Compaq_6710b_HW">
        <measurement type="pcr" name="3">FF017D...</measurement>
        <measurement type="pcr" name="5">86A000...</measurement>
```

```
        ...
      </report>
      <report type="tpm_pcrs" name="VDCnode_tGRUB_Xen">
        <measurement type="pcr" name="4">2944DD...</measurement>
        <measurement type="pcr" name="7">1E3F58...</measurement>
        ...
      </report>
    <reports>
    <systems>
      <system name="vdc_node" type="host_tcb" auth="VDC_nodes">
        <component_ref attest="TPM_1.2_default">HP_Compaq_6710b_HW</component_ref>
        <component_ref attest="TPM_1.2_default">VDCnode_tGRUB_Xen</component_ref>
      </system>
      <system name="remoteTVDnode" type="host_tcb" auth="VDC_nodes"> ... </system>
    </systems>
  </tvd_nodes>
  <tvd_layout>
    <reports>
      <report type="dboot_hvm" name="Webserver">
        <measurement type="digest" algo="sha1" name="config">9B659E...</measurement>
        <measurement type="digest" algo="sha1" name="kernel">AD3600...</measurement>
        <measurement type="digest" algo="sha1" name="initrd">42DD0B...</measurement>
        <measurement type="digest" algo="sha1" name="disk" id="0">FC59FF...</measurement>
      </report>
      <report type="full_hvm" name="MySQLDB"> ... </report>
      <report type="full_hvm" name="XenTVDmgmtVM"> ... </report>
    </reports>
    <resources>
      <resource type="network" name="mgmt_network">
        <encapsulation type="remote" mode="ipsec_tunnel_esp">
          <network_addr>10.0.2.0</network_addr>  <multicast_addr>10.0.2.255</multicast_addr>
          <cidr_suffix>24</cidr_suffix> <gateway>134.147.101.43</gateway>
          <ciphersuite>HMAC_SHA1_96_AES256_CBC</ciphersuite> <key type="psk">5d%f54Gs$82....</key>
        </encapsulation>
        <encapsulation type="local" mode="vlan"></encapsulation>
      </resource>
      <resource type="network" name="internal_network"> ... </resource>
      <resource type="network" name="InternetUplink"> ... </resource>
      <resource type="storage" name="PublicDocumentStorage">
        <file>file:///mnt/tvd/blue.tvd.opentc.net/resource_refs/docDB.img</file>
      </resource>
      <resource type="storage" name="MysqlDB_image">
        <encryption type="aes-cbc-essiv:sha256" key="mySecretStorageKey"/>
        <file>file:///mnt/tvd/blue.tvd.opentc.net/resource_refs/datarepo.img</file>
      </resource>
    </resources>
    <systems>
      <system name="XenTVDmgmtVM" type="vm" description="TVD Mgmt VM (Xen default)">
        <component_ref>XenTVDmgmtVM</component_ref>
        <resource_ref type="network" attach_id="1">mgmt_network</resource_ref>
      </system>
      <system name="Webserver" type="vm" description="Simple Xen WebServer">
        <component_ref>Webserver</component_ref>
        <resource_ref type="network" attach_id="0">InternetUplink</resource_ref>
        <resource_ref type="network" attach_id="1">internal_network</resource_ref>
        <resource_ref type="storage" attach_id="0">PublicDocumentStorage</resource_ref>
      </system>
      <system name="DatabaseServer" type="vm" description="MySQL DB Backend">
        <component_ref>MySQLDB</component_ref>
        <resource_ref type="network" attach_id="0">internal_network</resource_ref>
        <resource_ref type="storage" attach_id="1">MysqlDB_image</resource_ref>
      </system>
    </systems>
  </tvd_layout>
</tvd_policy>
```