Transparent Mobile Storage Protection in Trusted Virtual Domains

Luigi Catuogno[1]*, Hans Löhr*[1]*, Mark Manulis*[2]*, Ahmad-Reza Sadeghi*[1]*, Marcel Winandy*[1]
[1] *Horst Görtz Institute for IT Security*
Ruhr-University Bochum, Germany
{luigi.catuogno, hans.loehr, ahmad.sadeghi, marcel.winandy}@trust.rub.de

[2] *Technische Universität Darmstadt,*
Center for Advanced Security Research Darmstadt (CASED)
Germany
mark@manulis.eu

Abstract

Mobile Storage Devices, such as USB flash drives, offer a flexible solution for the transport and exchange of data. Nevertheless, in order to prevent unauthorized access to sensitive data, many enterprises require strict security policies for the use of such devices with the effect of rendering their advantages rather unfruitful.

Trusted Virtual Domains (TVDs) provide a secure IT infrastructure offering a homogeneous and transparent enforcement of access control policies on data and network resources, however, the current model does not specifically deal with Mobile Storage Devices.

In this paper, we present an extension of the TVD architecture to incorporate the usage of Mobile Storage Devices. Our proposal addresses three major issues: coherent extension of TVD policy enforcement by introducing architectural components that feature identification and management of transitory devices; transparent mandatory encryption of sensitive data stored on mobile devices; and highly dynamic centralized key management service. In particular we address offline scenarios allowing users to access and modify data while being temporarily disconnected from the domain. We also present a prototype implementation based on the Turaya security kernel.

Keywords: security, mobile storage devices, USB storage, trusted virtual domains

1 Introduction

Trusted Virtual Domains (TVDs) [22, 9] are the forthcoming framework for the implementation of multi-domain/single-infrastructure computer networks like centralized data centers, where computational resources from different owners share the same physical infrastructure, or single organizational LANs that span over different offices, branches or functional areas.

Amongst the strengths of TVDs is the transparent enforcement of access control policies — platforms and users logically assigned to the same TVD can access distributed data storage, network services, and remote servers without executing any additional security protocols, while the resources belonging to different TVDs are strictly separated and, thus, remain inaccessible.

In this paper, we extend the security concept of TVDs to capture the use of Mobile Storage Devices (MSDs) such as portable hard drives and USB sticks, which offer additional flexibility for the transport of data across multiple working locations and devices (e.g., work stations, printers, cell phones, cameras, etc.). The non-triviality of this task results from the diverse security risks with regard to the data stored on MSDs. For example, MSDs can be easily lost or stolen, and consequently the confidentiality of data becomes an issue. Once left unattended by the user, MSDs can be manipulated with the goal to breach the integrity of the data or to disseminate corrupted data or malicious code once the device is re-connected to the enterprise platform. Many security solutions for MSDs adopted in practice rely on a mixture of different techniques. In fact, the choice of appropriate mechanisms is guided by trade-off between their costs and offered benefits [33, 4]. Recent surveys indicate that existing security policies vary across organizations from none to very restrictive ones disallowing MSDs at all [16, 17, 45].

The deployment of MSDs is a challenging task for the current TVD model. Indeed, TVD infrastructures that want to take the major advantages of versatility of mobile storage devices have to address two main objectives: On the one hand, they should be *efficient* enough to reduce the overhead of enforcing security policies; on the other hand, they have to be *secure* enough to reduce the efforts requested to users and consequently reducing the effects of human errors.

Our Contribution In this paper we present an enhanced secure management model for MSDs within the framework of TVDs. Moreover, we present the design and the implementation of a comprehensive solution to enable transparent user-friendly encryption of sensitive data within a TVD. In particular, we address the usage of mobile storage devices to transport data within a domain by pursuing a separation between data storing and centralized key management. This separation is necessary to achieve offline data access, e.g., to allow a platform that is temporarily disconnected from the domain to process the data while preserving the desired security properties.

Paper organization We describe the problem definition in more detail in Section 2, and briefly overview the existing TVD concept in Section 3. Section 4 introduces our solution of integrating MSD management in TVDs, whereas we discuss the details of our MSD access control in Section 5. We describe our prototype implementation (Sec. 6), evaluate the security of our approach (Sec. 7), and discuss related work (Sec. 8). Section 9 concludes our work.

2 Problem Description

TVDs (see Section 3 for background) introduce a homogeneous and transparent infrastructure that aims at the separation between multiple domains with different security and trust policies. Enterprises and other organizations often have to deal with data of more than one security level. As a consequence, they separate their workflows to meet the different security requirements of their domains, e.g., working with confidential (internal) and public documents at the same time. The application of a TVD infrastructure can help these organizations to transparently enforce their security policies.

The incorporation and usage of mobile storage devices in TVDs would increase the flexibility of users in their workflows, but poses a challenging task in the design of the overall security architecture. MSDs are regularly employed to store copies of documents that the user may take home or to another office, raw data to be processed elsewhere, or on-the-fly data backups. In particular, MSDs are frequently used *offline*, i.e., plugged to any platform while it is not connected to the domain network (e.g., a laptop on the airplane).

MSD deployment raises several concerns about data confidentiality and integrity. Adversaries could intercept (steal) devices and try to read private data or even to make unauthorized changes. While encryption and digital signatures can achieve confidentiality and integrity of data stored on MSDs, the average human user is likely to be unskilled to properly configure and use standard security solutions. This may increase the probability of human errors and result in ineffective data protection. Moreover, users may feel any security policy as a nuisance that introduces overhead in their tasks and, therefore, try to circumvent or ignore it.

One important issue is that MSDs are passive components, thus enforcement of security policies relies on the computer they are connected to. We may assume the policy is correctly enforced as long as the MSDs are used within the TVD boundaries. This assumption is in general no longer true if any MSD is used outside its domain, e.g., when is connected to an outsider computer.

Our aim is to extend the TVD model with the benefits of using MSDs, allowing the transparent binding of an MSD to a certain TVD so that only platforms of the same TVD can access the stored data. Deploying MSDs within the TVD requires some refinement to the model due to the following concerns:

- *Device identification.* An MSD can move from a workstation to another without any control by the TVD infrastructure. Hence, whenever an MSD is plugged in, the platform should be able to distinguish the device and the domain this device belongs to.
- *Dynamic Device Management.* Unlike weighty storage devices, MSDs may unpredictably appear and disappear within the domain, according to the users' needs. This requires the introduction of an MSD management infrastructure in order to handle, e.g., creation and distribution of encryption keys.

2.1 The Offline Scenario

As mentioned above, MSDs are also used offline (i.e., the policy-enforcing platform is not connected to the domain), which introduces additional security problems. Almost all duties related to policy enforcement (e.g., authentication, key distribution, etc.) rely on interactive protocols. But policy rules may change, platforms may join/leave the domain (and should no longer access data), (disclosed) encryption keys may be revoked (and new ones should be generated and distributed). Whenever a policy change occurs, these changes have to be promptly propagated to all platforms in order to prevent further disclosure or sensitive data.

Hence, allowing offline platforms to access domain data stored on MSDs needs to fulfill the following security requirements:

- *Delegation*. Each domain platform should be able to enforce a policy (this means online and offline). For instance, each platform should store locally an instance of the policy and any credentials needed to enforce the policy.

- *Delayed revocation.* The notification of revocation of any platform, compartment, or device to offline platforms is delayed to the time they will re-connect to the domain network. In the meantime, data processed by these platforms and transferred over the domain through a mobile device may be made partially (or totally) invalid because of revocation. In order to validate data on mobile storage devices, every platform should be able to verify whether the data has been processed by a revoked platform.

- *Authentication and data integrity.* Access and data modification should be infeasible for outsiders.

- *Traceability and recovery.* Domain members should be able to track unauthorized data modifications and to reconstruct the previous data layout.

3 Background on Trusted Virtual Domains

Trusted Virtual Domains (TVDs) [22, 9] are a novel security framework for distributed multi-domain environments which leverages virtualization and trusted computing technologies. In this section we give a brief overview of the TVD concept and its features, and briefly introduce its main components and protocols.

In a virtualized environment, different applications and services together with their underlying operating systems are executed by different Virtual Machines (VMs) that share the same physical infrastructure. Each virtual machine runs in a logically isolated execution environment (which we call *compartment*), controlled by the underlying Virtual Machine Monitor (VMM). In such an environment, the user's work space is now executed by a virtual machine that is hosted by the VMM running on the physical platform along with other architectural components.

A TVD is a coalition of virtual machines that trust each other, share a common security policy and enforce it independently of the particular platform they are running on. Moreover, the TVD infrastructure contains the VMM and the physical components on which the virtual machines rely to enforce the policy. In particular, the main features of TVDs and the TVD infrastructure are:

- *Isolation of execution environments.* The underlying VMM provides containment boundaries to compartments from different TVDs, allowing the execution of several different TVDs on the same physical platform.

- *Trust relationships.* A TVD policy defines which platforms (including VMM) and which virtual machines are allowed to join the TVD. For example, platforms and their virtualization layers as well as individual virtual machines can be identified via integrity measurements taken during their start-up.

- *Transparent policy enforcement.* The Virtual Machine Monitor enforces the security policy independently of the compartments.

- *Secure communication channels.* Virtual machines belonging to the same TVD are connected through a virtual network that can span over different platforms and that is strictly isolated by the virtual networks of other TVDs.

Figure 1 shows an example of two TVDs that are distributed over different physical machines, and illustrates main components of the TVD architecture and their relations. The *TVD policy* is a set of rules that state security requirements a compartment should fit to be admitted to the TVD (e.g., integrity measurements of the platform and VMs) and defines both *intra*-TVD and *inter*-TVD information flow policy. A special node, namely the TVD Master, logically acting as a central server, controls the access to the TVD following the admission control rules stated in the TVD policy. The TVD Proxy is a compartment that locally enforces the TVD policy on the platform it is running on. Several TVDProxies, belonging to different TVDs can be instantiated on the same platform.

The process of TVD establishment in two steps, "deploy" and "join", is detailed in [29]: With the `TVD_deploy` protocol, the TVD Master verifies a platform and its ability to enforce the TVD policy. Then, in the `TVD_join` procedure, the TVD Proxy (verified by the TVD Master during the deploy phase) can verify virtual machines that are executed on the platform, and admit them to the TVD. Trusted computing technology is used to establish trust in the reported measurement values. For example – following the TCG approach – hash values of the software boot stack (including BIOS, bootloader, and virtualization layer as well as loaded virtual machines) are stored in and signed by a Trusted Platform Module (TPM) [43] and reported to the TVD Master during an attestation protocol. The TVD Master can reliably verify whether the reported values match the required ones of the TVD policy. Based on this, the TVD Master can implicitly rely on the enforcement mechanisms of the local platforms.[1]

Techniques to isolate and manage the virtual networks of different TVDs are given in [10]. Basically, virtual switches on each platform implement VLAN tagging for

[1]The definition of the required integrity measurement values in the TVD policy postulates knowledge about the behavior and security properties of the corresponding software programs. In practice, this can be achieved, e.g., through independent trusted third parties who evaluate and certify products according to evaluation standards like Common Criteria.

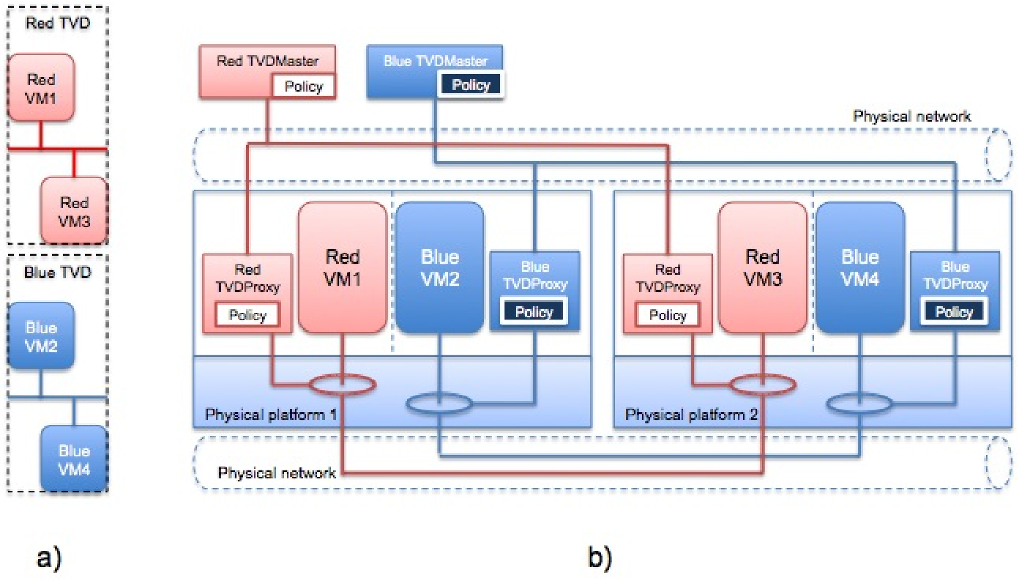

Figure 1: An overview of trusted virtual domains (TVDs)

Part a) shows the logical view of two TVDs, distributed over two physical platforms. Part b) shows the physical deployment of the TVD components, including the TVD Master .

local connections, and secure VPN for remote connections.

Various applications of TVDs were already shown and discussed in the literature. One example addresses the idea of applying the TVD concept for secure information sharing [26]. Other examples are virtual data centers [5], or enterprise rights management [20]. However, none of these works addresses the secure incorporation of mobile storage devices as we require.

3.1 Management of TVDs

The leading approach of management of TVDs within both centralized Virtual Data Centers and distributed organizational networks leverages on the deployment of advanced network management technologies (e.g., the *Web-based Enterprise Management* [14]) that provide highly integrated tools to accomplish administration tasks.

In a TVD-enabled infrastructure, management activities span over three levels. The *infrastructure level* concerns maintenance of physical resources, setup and configuration of the overall logical infrastructure, and assignment of resources to the different TVDs. At *domain level*, administrators take care of the TVD deployment, virtual machine setup and management of policies, devices and keys. Finally, at *compartment level*, running applications and current users can be notified of some events, coming from the underlying platform (e.g., revocation). At each level, administrators have an integrated management console that allows them to control all the operations under their responsibility. The administration of Virtual Data Centers with TVDs is discussed in [5].

4 Our Solution

In this section, we describe our solution to incorporate the use of mobile storage within the TVD framework. First, we describe how mobile storage operations are accomplished and, subsequently, we describe the new functionalities we introduce in the existing TVD architecture.

4.1 System Operation

Figure 2 shows an example TVD-enabled infrastructure in which two different TVDs are deployed. Each physical platform runs one or more virtual machines belonging to one of the existing TVDs. Several MSDs, variedly assigned to one of existing TVDs, are available to the users.

Here follows a typical usage example. The user Alice is working on the virtual machine VM_1 and plugs in her USB stick D_1 to the platform P_1 to make a backup copy of her files. Some specific components running on the platform P_1 (see Section 4.2.2) identify the plugged device, verify whether it has been assigned to the same TVD of VM_1 and retrieve the cryptographic keys that are used to encrypt and decrypt data on it. If everything succeeds, the device is made available to VM_1.

At this point, a further refinement to the device access control can be achieved on a per-VM basis. To this end, a set of rules that defines access privileges to each device assigned to the TVD (*device access policy*), has been added to the TVD policy. For each device, these rules state which operations and privileges (e.g., read, write) are granted to each virtual machine in the same TVD.

Hence, the platform P_1 allows VM_1 to mount the device D_1 under the constraints stated by the device access policy (read-only, read-write). Finally, if it is consistent with access privileges of VM_1, the copy of Alice's data can take place.

We recall that both device identification and key retrieval are performed automatically and transparently by the platform when the device is plugged in. The guest operating system of VM_1 does not need any special software to open and access the device, and no additional operation from the user (e.g., further authentications besides login, or providing keys) is required to handle data contained on the device. Moreover, we stress that data encryption is mandatory, thus the user cannot choose to not encrypt data once the mobile storage device has been assigned to a TVD.

Data stored on D_1 can be accessed only by those virtual machines which joined the same TVD. In particular, let D_1 be plugged in to platform P_3 which runs two virtual machines, VM_3 and VM_4. The virtual machine VM_3, which is in the same TVD as D_1, can access D_1, whereas VM_4 cannot. Trying to access D_1 on a virtual machine from a different TVD leads to a failure, because the platform is not allowed to retrieve the corresponding encryption key.

4.2 Virtual Storage Management

The main idea of our approach is to add access rules and the management of cryptographic keys for mobile storage devices at those components which are already responsible for handling access rules and keys for the TVD network, i.e., adding the information to the TVD policy and performing the enforcement by the TVD Master and the TVD Proxies. Moreover, we add additional components to the virtualization layer of each platform to deal with the specifics of MSDs: the *MSD Manager* and a *vMSD* component. Hence, the trusted components of virtual storage management in a TVD are the TVD Master and, on each platform, TVD Proxy, MSD Manager, vMSD, and, of course, the virtual machine monitor. We explain the interaction of these components in the following subsections.

4.2.1 Device Identification

Information needed for the identification of an MSD is contained in a special data structure named *identification record*, and stored on the device along with the data provided by the user. This information includes the name of the TVD the device belongs to, and the *device-id*, which uniquely identifies the device within the TVD. The identification record is generated and stored on the device when it is initialized.

4.2.2 Device Key Retrieval

To each MSD, our architecture associates a *security record* containing some security related information (see Section 5.2), including encryption keys. Security records of all MSDs are indexed by the *device-id* and are stored in a special database: the *Domain Device Directory (DDD)*, placed at the TVD Master. On every platform, each TVD Proxy handles a *Local Device Directory (LDD)* that partially replicates the domain directory of its domain. Physical platforms run a stand-alone component: the *MSD Manager*, which waits for a device to be plugged in. When this happens, the MSD Manager reads the device's identification record and extracts the *device-id* and the name of the TVD it is assigned to. The MSD Manager checks whether the TVD Proxy for that domain is running on the platform, and if so, the MSD Manager requests it to fetch the security record for the plugged device from the LDD.

If the record is found, the TVD Proxy allows the MSD Manager to open the device and releases its keys. If the TVD Proxy cannot find the requested record, it forwards the request to the TVD Master, which in turn searches for the record in the Domain Device Directory and replies either the requested record or an error message. Finally, the TVD Proxy stores the received record in the Local Device Directory and goes on.

The Local Device Directory fulfills two important functions. The first one is: allowing offline platforms to open a subset of mobile storage devices assigned to their domain, provided the corresponding security records have been added previously. The second one is: avoiding that the TVD Master is queried every time an MSD is used within its domain.

4.2.3 Accessing Devices

As stated above, neither the user, nor the virtual machine (which runs a commodity operating system) need to perform any additional task to access the MSD. Indeed, in our architecture, plugging in an MSD to a platform looks like plugging in a plain mobile storage that stores data in clear, from the virtual machine's point of view.

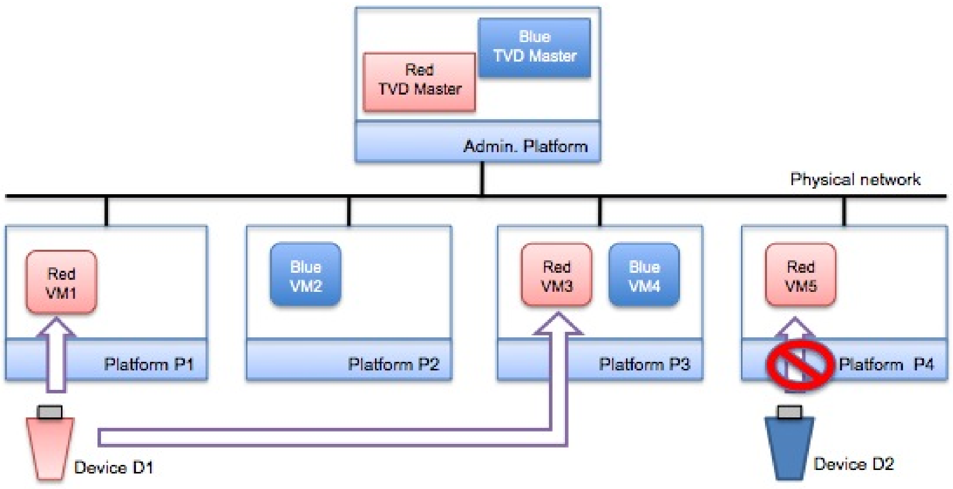

Figure 2: Example of using MSDs in an environment with two TVDs respectively named *red* and *blue*.

Data encryption (as well as device access policy enforcement) is performed by a specific component running on the platform: the *virtual MSD (vMSD)*. The vMSD features an encryption layer through which the VM mounts and accesses the device.

More precisely, a vMSD instance is created for each MSD plugged in to the platform and is given the corresponding keys by the TVD Proxy once the key retrieval has been completed successfully. Hence, the vMSD announces itself to the VM as a virtual device. All data the virtual machine reads/writes through the virtual device is silently processed by the vMSD layer and stored on the real MSD.

4.3 System Administration

4.3.1 Device Initialization

New mobile storage devices are assigned to a TVD through an initialization procedure. When an unassigned MSD is plugged in to a platform, the user is asked whether the system may initialize it.

The initialization requires the cooperation of the TVD Master. Indeed, the TVD Proxy running on the platform requires the TVD Master to generate the identification record and the security record (see Section 5.2) for the new MSD. The former is sent back to the platform and stored to the device via the vMSD whereas the latter is saved in the domain device directory and propagated to the requesting platform through the key retrieval procedure.

The device access policy can be determined at different levels. Users can explicitly provide the rules they need for their devices, or some general rules, stated both at platform or at domain level can be applied as default policy. Anyway, it is the TVD Master which writes the requested rules to the TVD policy.

When an MSD should be removed from a TVD, it can be de-initialized by simply deleting its security record from the Domain Device Directory (see Section 5.2).

4.3.2 Revocation

Any user, virtual machine or platform, may leave the TVD for administrative reasons or can be revoked because any kind of corruption has been discovered. In both cases, the administrator has to edit the TVD policy and any other involved data structures at the TVD Master (e.g. the Domain Device Directory).

Administrative revocations can be integrated within the setup and configuration procedures featured by the employed network management framework, so that, while modifying the layout of the network, administrators can consistently update the TVD policy.

The TVD architecture allows the TVD Master to realize whether platforms or virtual machines have been corrupted when they try to respectively deploy or join the TVD. The consequent failure can be notified to the administrator who can adopt the needed measures through the management facilities.

At the moment, the architecture presented in this paper does not feature any mechanism to automatically detect run-time intrusions. We discuss details of revocation in Section 5.5.

5 MSD Access Control Management

In this section we describe the realization of our MSD access control management. First, we briefly describe the enabling technologies, mainly cryptographic primitives we use, followed by a description of the initialization phase, and how the access control of MSDs is handled. Last but not least we present the more advanced feature of key revocation.

5.1 Building Blocks

In our architecture we apply two cryptographic primitives: a symmetric encryption scheme with lazy revocation for data encryption and an identity-based signature scheme for data authentication. Our solution is intended to be independent from the employed cryptographic primitive, so we base our design on a general model like the one discussed in [2]. Therefore, we briefly recall terminology and notation needed in the following. For more details, we refer to [2].

5.1.1 Lazy Revocation

A group of users share some data encrypted with the same symmetric encryption algorithm. In general, a validity time (timeslot) is assigned to each key. So, if t is the current timeslot, all keys k_i generated at times $i < t$, are considered revoked. At time t, all group members know the *current* key k_t. Whenever a user leaves the group, the current key is revoked and the new key k_{t+1} is generated and delivered to the remaining group members. The lazy revocation concept is based on the assumption that protecting old data from revoked users is not necessary since they could have accessed the data already and disclosed it to outsiders or other parties. Hence, previously encrypted data are not re-encrypted, whereas new data will be encrypted with the new key in order to preserve confidentiality. Anyway, each user still needs old keys, to read data encrypted at previous timeslots.

To avoid that participants store all revoked keys, several schemes [3, 32] provide users with a single *user master key* K_t for each timeslot t. K_t can be used to *extract* all keys k_i $(0 \leq i \leq t)$. This kind of schemes is characterized by a *trusted status* for each timeslot t. The initialization algorithm of the lazy revocation scheme generates the initial engine state E_0 related to the timeslot $t = 0$. User master key K_0 is *derived* from E_0. When a revocation occurs, the scheme *updates* its state taking current state E_t to the new state E_{t+1}, hence, a new master key K_{t+1} is derived and delivered. Revoked users still know K_t, but cannot use it to *extract* the new key k_{t+1}.

5.1.2 Identity-Based Signature

Let $W = \{w_1, \ldots, w_n\}$ be a group of identities (of users or platforms), represented as binary strings. An Identity-Based Signature (IBS) scheme [23, 18] is initialized by a trusted Key Generation Center (KGC) which generates the *master secret key* SK and the corresponding *master public key* PK. Then, using SK and an identity w, KGC can derive the appropriate *secret signing key* SK_w, which it then securely transports to w. This allows w to generate own signatures σ_w on any message of its choice, which can be verified by others using the identity w and the master public key PK.

5.2 Initialization

For each TVD, there is a TVD Master, which is assumed to be always online in order to handle new key retrieval requests from the various platforms. The TVD Master creates and manages for each mobile storage device the states E_t and master keys K_t for lazy revocation, as well as the master secret key SK and master public key PK for the identity-based signature scheme. To allow each platform to verify signatures made by the TVD Master, we assume a public-key infrastructure that enables the TVD Master to issue certificates for new master public keys.

In particular, the initialization ("coloring") of a new mobile storage device $\mathcal{D}$ for a TVD works as follows. Let the TVD be identified by (have the color) *tvdID*. Assume $\mathcal{M}$ to be the TVD Master of *tvdID*. Once the blank device $\mathcal{D}$ is connected to a platform $\mathcal{P}$, the Virtual Storage Management of $\mathcal{P}$ formats the device and requests the local TVD Proxy belonging to *tvdID* to generate an identification record IR for the device. The TVD Proxy contacts the TVD Master to issue the record containing a newly generated device-id d and *tvdID*. Figure 3 shows the corresponding protocol.

Beside the creation of the identification record, $\mathcal{M}$ also initializes encryption and signature schemes for $\mathcal{D}$. $\mathcal{M}$ creates the tuple (E_0, PK, SK, W, RL), where E_0 is the initial state of the symmetric encryption scheme, PK is the master public key and SK the master secret key for the IBS scheme, W is the set of writers (it is given as input to the initialization procedure) and RL, initially empty, is the set of revoked writers. All these information associated to $\mathcal{D}$ are stored in a newly created entry in the *Domain Device Directory* (DDD) on $\mathcal{M}$.

$\mathcal{M}$ derives its own signing key SK_M from SK. $\mathcal{M}$ signs the identification record $(d, \textit{tvdID})$ under SK_M and sends the result $IR := (d, \textit{tvdID}, sig[SK_M](d, \textit{tvdID}))$ to the TVD Proxy, which in turn stores it to the device. Now we have $\mathcal{D}.id = d$ and $\mathcal{D}.owner = \textit{tvdID}$. The latter indicates to which TVD the mobile storage device is assigned, i.e., the "color" of the TVD. Note that storing files from different TVDs on the same device is logically equivalent to having one device for each TVD. Here, for simplicity, we consider only the second case.

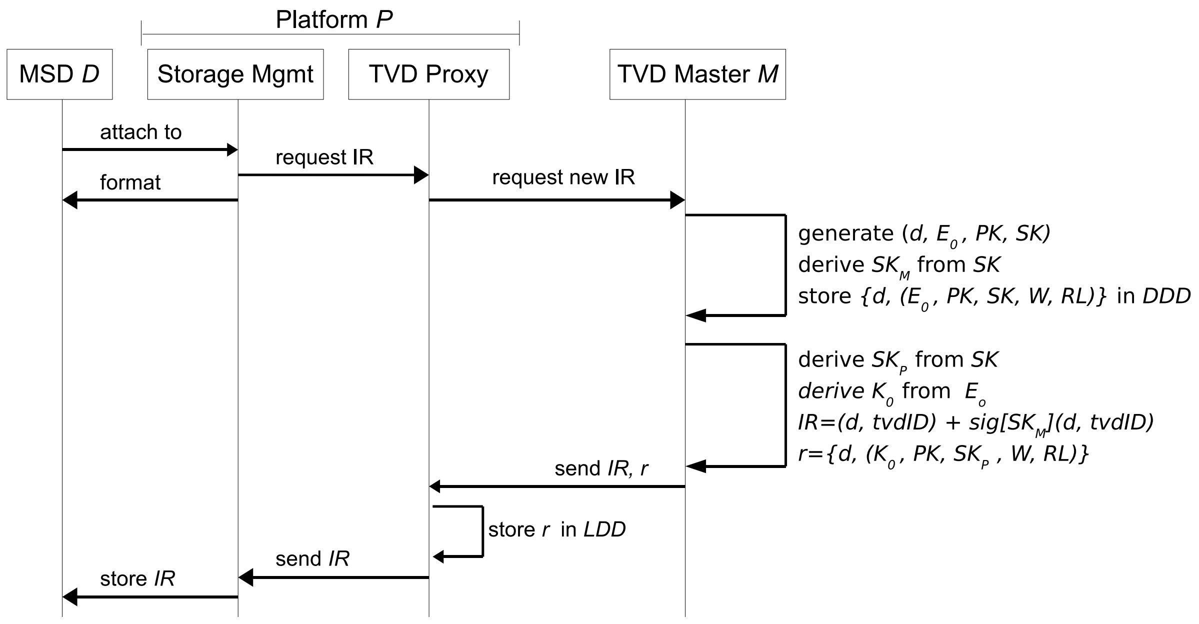

Figure 3: Device coloring protocol.

Note that neither E_i nor SK are delivered to any platform, they are stored and processed only on the TVD Master $\mathcal{M}$. Indeed, $\mathcal{M}$ distributes to each platform $\mathcal{P}$ the key management record $r = (d, (K_t, PK, SK_P, W, RL))$ where: K_t is the *current* master key for encryption, and SK_P is the signing key of the platform $\mathcal{P}$ which was derived from SK by the TVD Master. The record r is stored in the *Local Device Directory* (LDD) of the corresponding TVD Proxy on $\mathcal{P}$.

Device De-Initialization Finally, a device can be "uncolored" by deleting its identification record (by formatting it) and erasing its corresponding entries in the global (DDD) and local device directories (LDD). Entries in both directories can have an expiration time, to avoid that the TVD Master keeps information about devices forever.

5.3 MSD Access Control Mechanism

When a device $\mathcal{D}$ assigned to the TVD is attached to the platform $\mathcal{P}$, which hosts VMs of the same domain, then the Virtual Storage Management of $\mathcal{P}$ extracts the identification record IR from the device. If the device is recognized, i.e., $\mathcal{D}.owner$ is this TVD and the signature of IR is valid, then the MSD Manager requests the corresponding TVD Proxy to search for the record indexed by *d*=$\mathcal{D}$*.id* in its Local Device Directory in order to obtain the device keys. If the entry is not found because the device has not been attached to this platform yet before, the query is forwarded to the TVD Master $\mathcal{M}$.

5.4 File Storage

In a naive approach, a platform, once it has obtained required keys, gets the whole file from the device, decrypts it, verifies the attached signature and makes it available to user applications. Eventually, it encrypts and signs the updated file and copies it back to the device. This approach raises a problem: new data overwrite old data.

To fulfill our *traceability and recovery* requirement, we store data to the MSD using a *versioning filesystem*.

In a versioning filesystem, each file can exist in several versions. Usually, users can access transparently (unlike in conventional application-level revision control systems [41]) the latest file version as in a regular filesystem, whereas a set of user-level utilities feature several administrative tasks on file versions.

Versioning filesystems allow to record the history of changes to files in data repositories, and are useful where it is needed to maintain accurate logs of data flows and possibly to reverse some operations.

The versioning policy we adopted is known as *Copy-on-write*: a new file version is created each time it is modified, e.g., by a write operation. Hence, a node which accesses any input file f_i, saves (and signs) its version to the mobile storage device as a new file f_{i+1}, instead of simply overwriting the previous one. Afterwards, each node can load the latest version of any file for which it can successfully verify the signature.

As a consequence of this versioning policy users will progressively consume all the available storage space on any device. Therefore, the TVD policy also defines a *purge* privilege that allows certain users to delete or merge old versions. The purge operation is rather critical, hence, it is intended to be done only by domain administrators and only when the device is connected to an online platform.

We embedded both data security (encryption and digital signatures) and handling file revisions into our architecture. The Virtual Storage Management performs the corresponding operations transparently with respect to user compartments.

5.5 Revocation of Cryptographic Keys

Both encryption and signing keys can be revoked in three cases:

- *Member revocation:* Whenever a platform, VM, or user is no longer member of the domain, the TVD Master updates the encryption key (and revokes the signing key if any).
- *Key disclosure:* Whenever it is known that a key has been disclosed to unauthorized parties (e.g., due to malicious users or compromised platforms), the corresponding key must be revoked.
- *Expiration:* Creating and updating keys are bound to a timer.

Suppose that at time t, revocation of k_t is requested, $\mathcal{M}$ updates the encryption engine taking it from state E_t to state E_{t+1}, derives the new master key K_{t+1}. K_{t+1} is delivered to platforms that can extract the new encryption key k_{t+1}.

To revoke the signing key SK_w, the TVD Master $\mathcal{M}$ adds w to the revocation list RL. If the revoked key has to be replaced by a new one, $\mathcal{M}$ generates a new writer-id w', puts it into the set W of write-enabled nodes and sends it to the node previously known as w. Moreover, $\mathcal{M}$ sends the new revocation list RL to all other platforms. All data signed with the revoked key SK_w are no longer accepted by any platform.

Key revocation may occur asynchronously with respect to device access and the periodical update requests by TVD members. Therefore we setup a key event notification system in which the TVD Master notifies a key revocation to all platforms hosting VMs of the domain by raising an appropriate event or alarm. Once the event notification has been received, each online platform renews its keys. Event notifications are queued, so that they can be delivered to offline platform once they connect to the TVD.

5.6 Offline Scenario

We briefly revisit how requirements raised in the offline (but also online) scenario are addressed by our MSD access control management:

- *Delegation*: Once the `TVD_deploy` protocol [29] has been carried out, the TVD Proxy locally stores an instance of the TVD policy and a certain set of MSD key management records. Hence, it is allowed to enforce the policy and guarantee the access to the subset of MSDs whose keys are stored in the Local Device Directory LDD.
- *Lazy revocation*: Whenever a key revocation occurs, new data, encrypted with the newly generated key, do not overwrite the previous ones, hence, the old data are still available for offline platforms to which the new key has not been delivered.
- *Authentication and integrity* are provided by the identity based signature scheme. Data written to a mobile storage device is digitally signed with the key assigned to the platform the device is attached to. Unauthorized changes afterwards can easily be detected by verifying the signature.
- *Traceability and recovery*: Employing a versioning file service allows to keep track of all modification made to the data, enabling offline platforms to access to the most recent version they can decrypt. Moreover, whenever a revocation occurs, it is possible to retrieve and delete all changes performed by the revoked platform.

6 Implementation

In this section we briefly describe our prototype implementation of the MSD management in a TVD architecture. In particular, we describe our implementation of the virtual storage management. Figure 4 illustrates our implementation.

Our prototype implementation is based on the Turaya [15] security kernel. Turaya is composed of two layers. The first layer is built upon an L4 microkernel [28], which ensures separation among logical execution environments (compartments) and runs services that feature resource management (memory management, I/O). On top of the L4 microkernel, compartments can be native processes (called L4 tasks) or virtual machines (e.g., L4Linux, which is a para-virtualized Linux). The trusted software layer provides security services including secure storage, compartment management, and trusted channel establishment.

In particular, a *trusted channel* [21, 39, 19, 1] is a secure channel established between two compartments

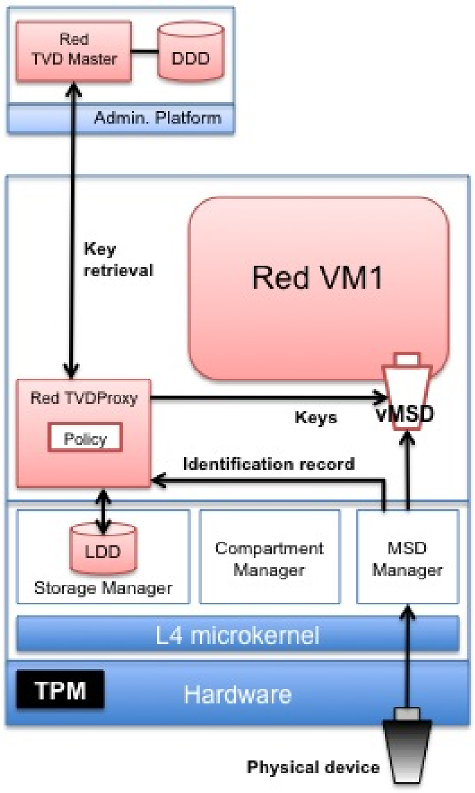

Figure 4: Compartments involved in mounting and accessing to an attached MSD.

that: (a) validate each the configuration and identity of the other, (b) negotiate each an encryption key depending on the configuration of the other, so that data sent over the channel by the former can be accessed only by the latter and *vice versa*. In our architecture, trusted channels are used for the communication between the TVD Master and TVD Proxies. In this way, the TVD Master is ensured that the TVD Proxy to which it is going to send security-critical data, is trusted, and that only that particular TVD Proxy can access data. On the other hand, the TVD Proxy is guaranteed that it is receiving data from the legitimate TVD Master. The *storage manager* is a compartment that provides a persistent storage for other compartments. The storage manager ensures confidentiality, integrity, and freshness of stored data and bind them to the configuration of their owner compartment. For binding data and establishing trusted channels we use trusted computing functionality of a TPM [43].

In addition to these services, our architecture introduces two components that allow the user compartments to access files on devices without having to deal with data encryption themselves.

- The MSD Manager is a compartment, implemented as an L4Linux virtual machine, which handles pluggable USB devices. When a USB disk device is attached, the MSD Manager reads the identification record of the device and forwards it to the TVD Proxy for device recognition and, hence, for key retrieval.
- The *Virtual MSD (vMSD)* maps the physical encrypted device to a virtual (clear) one which is made available to each user compartment of the corresponding TVD. Once the TVD Proxy has retrieved the device keys, it requests the Compartment Manager to create the vMSD, which is implemented as a native L4 task for each MSD. When the device is detached, the vMSD is deleted.

Turaya components plus TVD Master, TVD Proxy, MSD Manager and vMSD compose the Trusted Computing Base of our architecture.

As already exposed in Section 5.4, our architecture stores data on mobile devices through a secure versioning filesystem in which file content is encrypted and each file version is signed by the user that created it. In our early experiments, the versioning filesystem has been implemented on Linux as a filesystem layer based on the FUSE [40] module, making the system independent from the underlying device filesystem format.

The diagram in Figure 6 summarizes the performance measurement performed with the Bonnie benchmark tool [8]. The first column of each group of measurements (version only) shows the respective performance achieved by the filesystem without performing neither encryption nor signature of data. The values summarized by the second column are obtained by running the test enabling cryptographic features on single versioned files. The purpose of this test is evaluating the overhead introduced by the encryption scheme. In the third column we can observe the performance measurement on files that have eight previous versions. In this case, we aimed to point out the overhead due to the signature verification of the previous file versions.

We preliminarily point out that in general performance of devices like USB sticks, memory cards and so on are not very good if compared with the traditional hard disks and that the use of digital signatures (usually avoided in handling contents in secure filesystems) heavily affects performance. On the other hand, we notice that actually mobile devices are mostly used as temporary storage. That is, users prefer to copy documents lying on the pen drive to the local hard drive, edit them there, and then copy them back to the mobile device. Hence, in this usage scenario filesystem performance are not a critical issue. Nevertheless, our tests allowed us to focus on the filesystem reliability, though we find that these results are rather promising and some improvements could be achieved by enhancing the file signature mechanism.

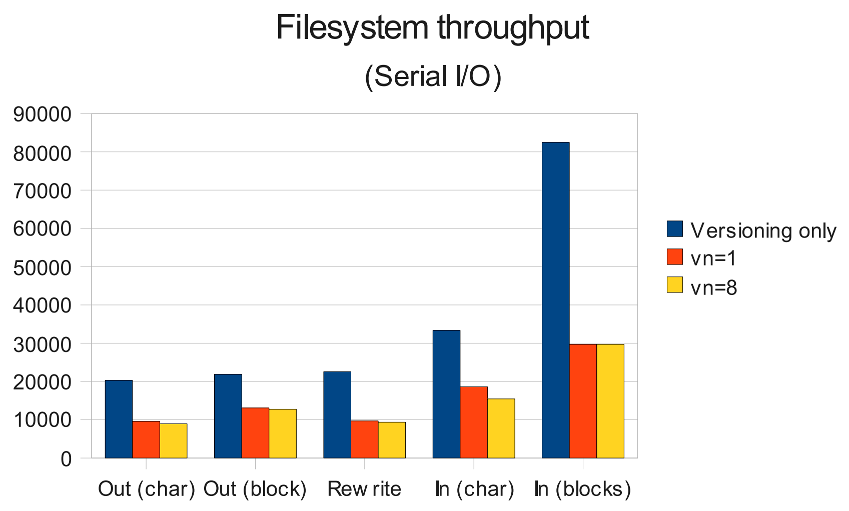

Figure 5: Performance measurement summary. The first column (versioning only) shows the throughput of the filesystem without cryptographic features. The remaining columns show test results on encrypted and signed files, where in average *Vn* versions are present.

7 Security Considerations

In this section we briefly focus on the main security aspects of our architecture, focusing on possible attacks an adversary could launch to untrusted components of our model: user application, devices and offline platforms.

We denote as "adversary" both an outsider entity who is unaware of any information about the network and its TVDs (e.g., old data and keys, namespace conventions etc.), and revoked members with insider information who are no longer trusted and are assumed to act as adversaries.

We assume that adversaries are not able to attack the platform hardware. For example, adversaries cannot examine the content of the memory and possibly extract any information by rebooting the platform and analyzing its memory [24].

Moreover, our scheme aims at preventing disclosure of sensitive data through the misuse of mobile storage devices, whereas it is possible, for example, that insider attackers could make an illegal copy of a confidential document by taking a picture of the screen with a camera, or simply printing it. However, in this work, we do not cope with this kind of threats.

- *Attacks to user applications*. An adversary who exploits a user application running in any user compartment (or the compartment's OS itself), enjoys his victim's access privileges to the plugged MSD. The adversary can obtain and modify data through the application environment. However, the attacker can neither obtain the MSD keys he is using nor can he force the platform to change the keys currently used, as keys and encryption algorithms are not present in the application environment. This threat could be mitigated by letting the TVD Proxy periodically verify the integrity of the user compartment (e.g., each time check if records in the *LDD* are up-to-date). If the verification fails, the TVD Proxy asks the Compartment Manager to kill the hijacked compartment and requests the TVD Master for revocation of the key of the MSD it was using.

- *Attacks to the MSD*. The MSD is a passive device that cannot enforce any effective security measure. An adversary who physically accesses the device, can copy, corrupt and delete both files and the identification record, making the device content no longer available to legitimate users. However, adversaries cannot forge any valid data signature since they have not any valid signing key. Moreover, adversaries cannot read the content of the files stored on the MSD, because they do not possess the necessary encryption keys. However, revoked users can still read data that is encrypted with a key they obtained before being revoked. Roll-back attacks are always possible. Adversaries could make a copy of the filesystem at a certain time and afterwards they can overwrite each more recent version of any file. Revoked users can roll-back the MSD content to a date it was still legitimate and begin an unauthorized branch of the data.

- *Attacks to offline platforms*. Data modifications carrying signatures by revoked users are no longer considered valid. Offline platforms may still successfully verify signatures by certain revoked users, and even produce some output based on possibly malicious input. However, when the MSD containing such "corrupted data" is connected to an online platform, unauthorized modifications can be found and discarded. Nevertheless, in this way, revoked members might still obtain new data from offline platforms to which the new key has not been delivered yet. Currently, our implementation does not include a solution to this problem.

8 Related Work

The widespread use of Mobile Storage Devices (e.g., memory cards, USB sticks, transportable solid-state hard disks), that allow users to move files among different workstations, poses several problems, *in primis* related to data confidentiality and integrity. In order to cope with these problems, cryptographic mechanisms, i.e., encryption and digital signatures are useful means.

Cryptographic filesystems [7, 12, 27, 25] embed encryption mechanisms into the filesystem operation, featuring a way to encrypt data and metadata without any effort by user level applications. This makes it possible to have good performance and fine-grained security. In particular, the Plutus filesystem [25] features lazy re-encryption [2] at the level of single file-blocks and the *key rotation* mechanism to efficiently generate and manage new encryption keys.

Traditionally, cryptographic filesystems provide a client-server architecture in which the former is trusted and features file content encryption, integrity verification and key management and the latter (untrusted) simply acts as storage for encrypted files. Although several encrypted filesystem can be used also to encrypt local storage devices, they best fit the networked scenario.

Solutions that focus on local storage encryption vary between full disk encryption enforced by hardware or software security modules and creating encrypted partition on local devices [30, 42]. In this case, the aim is guaranteeing the data confidentiality even if the device is stolen and connected to another computer.

The Virtual Private File System [44] leverages on virtualization to assure confidentiality whereas data is accessed through a possibly compromised operating system. Sensitive applications run in a trusted compartment and access their own separated storage through a filesystem layer that features data secrecy, integrity and recoverability and relies on the untrusted filesystem provided by a virtualized legacy operating system.

Encrypted filesystems as those mentioned above are built on top of a specific operating system and are generally not portable. This may introduce inacceptable constrains in a large scale environment. Moreover, distributed encrypted filesystems have, in many cases, their own key management infrastructure which may not be easily interoperable with other existing infrastructures (e.g., PKIs, LDAP). This introduces some redundancies and administrative overhead. Conversely, local storage encryption facilities essentially protect personal devices and workstation and do not feature any distributed key management service. The VPFS also suffers from this shortcoming. In contrast, our solution works for a wide range of applications and operating systems due to the virtualization approach. In fact, any application that can run in a VM transparently benefits from the underlying encryption mechanism. Moreover, it is possible to use the same mobile storage device with its encrypted data on various heterogeneous platforms since the TVD infrastructure provides an abstraction of the underlying encryption mechanism and its key management.

Several architectures aim at enforcing sophisticated security policies within large scale and multi-domain environments and are built on top of a filesystem encryption layer. In particular, the Concord framework [37] allows organizations to monitor data while it is accessed by mobile equipment and makes it possible to enforce the access policies even in a *disconnected* scenario. Institution's data are stored in encrypted form and encryption keys are shared (through a threshold encryption scheme) by a trusted policy enforcer and the user mobile device (e.g., a laptop). In order to access data, the user and the enforcer have to cooperate in order to reconstruct the data encryption key. This approach allows the infrastructure to promptly deny the access to data if it realizes the client has been compromised. In the disconnected scenario, the infrastructure restricts the user privileges to read-only accesses to a subset of organizational data. The role of the enforcer is played by a "disconnected" policy enforcer to which only a limited subset of encryption key shares has been delivered. To the best of our knowledge, Concord is the approach closest to our proposal. In our architecture, Concord's user machine and policy enforcer are collapsed into the same platform, though as different compartments, namely the virtual machine and the TVD Proxy. However, our solution features a less restricted off-line scenario (Concord's disconnected mode does not allow users to modify protected data). Moreover, the virtual storage management in TVDs is in general more flexibile and transparent to the user.

Traceability and reversibility of data modification is an important feature when allowing full data access within the offline scenario and can be achieved through so-called file versioning services, available both at application level [41, 6] and at filesystem level [13, 31, 38, 34]. In particular, several recent proposals address security and integrity checks for stored data, as well as verifiable audit trails [36, 11, 35]. However, these systems do not fit our requirements since they have not been designed to handle totally passive storage devices.

9 Conclusion and Future Work

In this paper we presented an architecture for the secure and transparent deployment of Mobile Storage Devices (MSDs) within Trusted Virtual Domains (TVDs). We believe that multi-domain IT infrastructures addressed in the TVD model take advantage from the use of these devices. We argued that the usually adopted approaches to protect data stored on these devices suffer from several shortcomings, mainly due to: their intrinsic untraceability, their lack of any effective security feature, and the considerable overhead that their management introduce into the users' work. Moreover, we pointed out that introducing MSDs within TVDs is not a trivial task if the resulting architecture should still be compliant with some typical usage scenarios of these devices, and in particular in the offline scenario. We introduced a general model of a multi-domain environment that enables the secure and transparent use of MSDs and showed how to extend existing TVD architectures with MSD management components to realize this model. Finally, we sketched a proof of concept implementation based on the Turaya security kernel, which uses an L4 microkernel to provide protected execution environments for management services and virtual machines for reusing user applications.

Another important security problem related to MSDs is the proliferation of malicious software like trojan horses and viruses. Our solution limits such attacks, because only data written and signed by legitimate members of a TVD is accepted as input by other TVD members. However, legitimate members might still spread malicious code inadvertently (e.g., if they are infected by a virus). Future research might be directed towards preventing the execution and propagation of malicious code from MSDs.

References

[1] Armknecht, F., Gasmi, Y., Sadeghi, A.-R., Stewin, P., Unger, M., Ramunno, G., and Vernizzi, D. An efficient implementation of trusted channels based on OpenSSL. In *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing (STC 2008)* (2008), ACM Press, pp. 41–50.

[2] Backes, M., Cachin, C., and 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), pp. 1–11.

[3] Backes, M., Cachin, C., and Oprea, A. Secure key-updating for lazy revocation. In *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006, Proceedings* (2006), vol. 4189 of *Lecture Notes in Computer Science*, Springer, pp. 327–346.

[4] Beautement, A., Coles, R., J., Ioannidis, C., Monahan, B., Pym, D., Sasse, A., and Wonham, M. Modeling the human and technological costs and benefits of USB memory stick security. In *Workshop on the Economics of Information Security (WISE)* (2008).

[5] Berger, S., Cáceres, R., Pendarakis, D. E., Sailer, R., Valdez, E., Perez, R., Schildhauer, W., and Srinivasan, D. TVDc: Managing security in the trusted virtual datacenter. *Operating Systems Review 42*, 1 (2008), 40–47.

[6] Berliner, and Polk. Concurrent versions system (cvs), 2001. `http://www.cvshome.org/`.

[7] Blaze, M. A cryptographic file system for unix. In *ACM Conference on Computer and Communications Security* (1993), pp. 9–16.

[8] Bray, T. Bonnie - filesystem benchmark tool. `http://www.textuality.com/bonnie`.

[9] Bussani, A., Griffin, J. L., Jansen, B., Julisch, K., Karjoth, G., Maruyama, H., Nakamura, M., Perez, R., Schunter, M., Tanner, A., Doorn, L. V., Herreweghen, E. A. V., Waidner, M., and Yoshihama, S. Trusted Virtual Domains: Secure foundations for business and IT services. Tech. Rep. RC23792, IBM Research, 2005.

[10] Cabuk, S., Dalton, C. I., Ramasamy, H. V., and Schunter, M. Towards automated provisioning of secure virtualized networks. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007* (2007), ACM, pp. 235–245.

[11] Cachin, C., and Geisler, M. Integrity Protection for Revision Control. In *Applied Cryptography and Network Security: 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009, Proceedings* (2009), Springer, p. 382.

[12] Cattaneo, G., Catuogno, L., Sorbo, A. D., and Persiano, P. The design and implementation of a transparent cryptographic file system for unix. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, June 25-30, 2001, Boston, Massachusetts, USA* (2001), USENIX, pp. 199–212.

[13] Cornell, B., Dinda, P., and Bustamante, F. Wayback: A user-level versioning file system for linux. In *Proceedings of Usenix Annual Technical Conference, FREENIX Track* (2004), pp. 19–28.

[14] Distributed Management Task Force. "web-based enterprise management (wbem)". `http://www.dmtf.org`.

[15] European Multilaterally Secure Computing Base(EMSCB) Project. Towards trustworthy systems with open standards and trusted computing, 2008. `http://www.emscb.de/`.

[16] European Network and Information Security Agency (ENISA). Secure USB Flash Drives, June 2008. `http://www.enisa.europa.eu/doc/pdf/publications/SecureUSBdrives_180608.pdf`.

[17] Fabian, M. Endpoint security: managing USB-based removable devices with the advent of portable applications. In *InfoSecCD '07: Proceedings of the 4th annual conference on Information security curriculum development* (New York, NY, USA, 2007), ACM, pp. 1–5.

[18] Galindo, D., Herranz, J., and Kiltz, E. On the generic construction of identity-based signatures with additional properties. In *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings* (2006), vol. 4284 of *Lecture Notes in Computer Science*, Springer, pp. 178–193.

[19] GASMI, Y., SADEGHI, A.-R., STEWIN, P., UNGER, M., AND ASOKAN, N. Beyond secure channels. In *Proceedings of the 1st ACM Workshop on Scalable Trusted Computing (STC'07)* (2007), ACM Press, pp. 30–40.

[20] GASMI, Y., SADEGHI, A.-R., STEWIN, P., UNGER, M., WINANDY, M., HUSSEIKI, R., AND STÜBLE, C. Flexible and secure enterprise rights management based on trusted virtual domains. In *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing, STC 2008, Alexandria, VA, USA, October 31, 2008* (2008), ACM, pp. 71–80.

[21] GOLDMAN, K., PEREZ, R., AND SAILER, R. Linking remote attestation to secure tunnel endpoints. In *Proceedings of the First ACM Workshop on Scalable Trusted Computing (STC'06)* (2006), pp. 21–24.

[22] GRIFFIN, J. L., JAEGER, T., PEREZ, R., SAILER, R., VAN DOORN, L., AND CÁCERES, R. Trusted Virtual Domains: Toward secure distributed services. In *Proceedings of the 1st IEEE Workshop on Hot Topics in System Dependability (HotDep'05)* (June 2005).

[23] GUILLOU, L. C., AND QUISQUATER, J.-J. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings* (1990), vol. 403 of *Lecture Notes in Computer Science*, Springer, pp. 216–231.

[24] HALDERMAN, J. A., SCHOEN, S. D., HENINGER, N., CLARKSON, W., PAUL, W., CALANDRINO, J. A., FELDMAN, A. J., APPELBAUM, J., AND FELTEN, E. W. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM 52*, 5 (2009), 91–98.

[25] KALLAHALLA, M., RIEDEL, E., SWAMINATHAN, R., WANG, Q., AND FU, K. Plutus: Scalable secure file sharing on untrusted storage. In *Proceedings of the FAST '03 Conference on File and Storage Technologies, March 31 - April 2, 2003, Cathedral Hill Hotel, San Francisco, California, USA* (2003), USENIX.

[26] KATSUNO, Y., KUDO, M., PEREZ, P., AND SAILER, R. Towards Multi-Layer Trusted Virtual Domains. In *The 2nd Workshop on Advances in Trusted Computing (WATC 2006 Fall)* (Tokyo, Japan, Nov. 2006), Japanese Ministry of Economy, Trade and Industry (METI).

[27] LI, J., KROHN, M. N., MAZIÈRES, D., AND SHASHA, D. Secure untrusted data repository (sundr). In *OSDI* (2004), pp. 121–136.

[28] LIEDTKE, J. On micro-kernel construction. In *SOSP* (1995), pp. 237–250.

[29] LÖHR, H., SADEGHI, A.-R., VISHIK, C., AND 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* (2009), vol. 5451 of *Lecture Notes in Computer Science*, Springer, pp. 396–407.

[30] MICROSFOT CORP. Bitlocker drive encryption, 2006. `http://technet.microsoft.com/en-us/windows/aa905065.aspx`.

[31] MUNISWAMY-REDDY, K., WRIGHT, C. P., HIMMER, A., AND ZADOK, E. A Versatile and User-Oriented Versioning File System. In *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST 2004)* (San Francisco, CA, March/April 2004), USENIX Association, pp. 115–128.

[32] NAOR, D., SHENHAV, A., AND WOOL, A. Toward securing untrusted storage without public-key operations. In *Proceedings of the 2005 ACM Workshop On Storage Security And Survivability, StorageSS 2005, Fairfax, VA, USA, November 11, 2005* (2005), ACM, pp. 51–56.

[33] PARKIN, S. E., KASSAB, R. Y., AND VAN MOORSEL, A. P. A. The impact of unavailability on the effectiveness of enterprise information security technologies. In *Service Availability, 5th International Service Availability Symposium, ISAS 2008, Tokyo, Japan, May 19-21, 2008, Proceedings* (2008), vol. 5017 of *Lecture Notes in Computer Science*, Springer, pp. 43–58.

[34] PETERSON, Z., AND BURNS, R. Ext3cow: a time-shifting file system for regulatory compliance. *ACM Transactions on Storage (TOS) 1*, 2 (2005), 190–212.

[35] PETERSON, Z., BURNS, R., ATENIESE, G., AND BONO, S. Design and implementation of verifiable audit trails for a versioning file system. In *Proceedings of the 5th USENIX conference on File and Storage Technologies table of contents* (2007), USENIX Association Berkeley, CA, USA, pp. 20–20.

[36] SHAPIRO, J. S., AND VANDERBURGH, J. Access and integrity control in a public-access, high-assurance configuration management system. In *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002* (2002), USENIX, pp. 109–120.

[37] SINGARAJU, G., AND KANG, B. H. Concord: A secure mobile data authorization framework for regulatory compliance. In *Proceedings of the 22nd Large Installation System Administration Conference, LISA 2008, November 9-14, 2008, San Diego, CA, USA* (2008), USENIX Association, pp. 91–102.

[38] SOULES, C., GOODSON, G., STRUNK, J., AND GANGER, G. Metadata Efficiency in Versioning File Systems. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies* (2003), USENIX Association Berkeley, CA, USA, pp. 43–58.

[39] STUMPF, F., TAFRESCHI, O., RÖDER, P., AND ECKERT, C. A robust integrity reporting protocol for remote attestation. In *2nd Workshop on Advances in Trusted Computing (WATC'06 Fall)* (Tokyo, December 2006).

[40] SZEREDI, M. File system in user space. `http://sourceforge.net/projects/fuse`.

[41] TICHY, W. Design, implementation, and evaluation of a Revision Control System. In *Proceedings of the 6th international conference on Software engineering* (1982), IEEE Computer Society Press Los Alamitos, CA, USA, pp. 58–67.

[42] TRUECRYPT FOUNDATION. Truecrypt - free open-source on-the-fly encryption, 2004. `http://www.truecrypt.org/`.

[43] TRUSTED COMPUTING GROUP. TPM main specification, version 1.2 rev. 103, July 2007. `https://www.trustedcomputinggroup.org`.

[44] WEINHOLD, C., AND HÄRTIG, H. VPFS: Building a virtual private file system with a small trusted computing base. In *Proceedings of the 2008 EuroSys Conference, Glasgow, Scotland, UK, April 1-4, 2008* (2008), ACM, pp. 81–93.

[45] WIRED. Under worm assault, military bans disks, USB drives, Nov. 2008. `http://www.wired.com/dangerroom/2008/11/army-bans-usb-d`.