Secure Virtual Disk Images for Grid Computing

Carl Gebhardt
Information Security Group
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
c.gebhardt@rhul.ac.uk

Allan Tomlinson
Information Security Group
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
allan.tomlinson@rhul.ac.uk

Abstract

We present in our paper a secure, flexible and transparent security architecture for virtual disk images. Virtual disk images are often overlooked in security concepts, especially in a grid environment where disk images are considered to be secure as long as they reside within the secured borders of the data center. However, for some applications this level of assurance is not satisfactory. In our security architecture, virtualized guests transparently benefit from integrity as well as confidentiality assurance. Traditional virtual disk images lack the ability of an efficient integrity protection mechanism. We base our concepts on trusted computing utilizing the Trusted Platform Module (TPM) to efficiently deliver integrity assurance to virtual disk images. Further, we allow a restrictive rule-set to be imposed by the virtual disk image owner, and we enable the owner to retain control over the virtual disk image throughout its life-cycle.

1 Introduction

The idea of running several operating systems simultaneously on the same machine is not new, having emerged in the early 1960's, after its basic concepts and ideas were presented in 1959 [11]. However, as the cost of computing systems falls and performance increases, the idea of virtualization has found renewed interest and is now a growing field of commercial interest as well as a popular area of academic research [5, 3].

It is not surprising then, that virtualization is becoming increasingly popular in the context of grid computing [5, 1]. This is due to the advantages it offers in management, utilization and flexibility. Furthermore, it allows heterogeneous operating systems (OS) and applications to run on the same machine simultaneously, while providing strong isolation between them.

Although virtualization offers many benefits to users, new security concerns and special issues are becoming apparent [3, 4]. Solving these security problems will require a sophisticated architecture based upon sound security principles and the development of secure modules. Securing virtual disk images is one particular aspect of this research. One threat, for example, is that virtual disk images can be modified without the legitimate user's knowledge. Moreover, such an image could contain sensitive data and security credentials which may enable other threats [3, 4]. With this paper therefore, we intend to address the special issues of virtual disk images and propose our vision of secure virtual disk images (SVDI).

In general, virtual disk images (VDI), also known as virtual hard disks, represent the hard drive of a virtualized guest operating system encapsulated in a single file. The disk image file can contain a whole operating system, data and/or applications. For the guest system this also means that it is irrelevant whether the disk image is located on the host's hard-drive, mounted via a network share or hosted over the Internet. The location of the image and its access method is transparent for the guest. For the guest system this poses a new set of security challenges - it cannot make any assumption about the integrity nor the confidentiality of its own hard drive content. Data could be altered deliberately or accidentally during storage or transit without the knowledge of the guest. To some extent this threat may be mitigated by cryptographically protecting the integrity of individual files within the guest operating system. However, not all applications employ cryptographic technology and often assume that the execution environment itself is secure.

Other applications rely on non-keyed hash functions or checksums to protect integrity, assuming that attackers do not have access to the integrity metric itself. Depending on the specific circumstances these may be reasonable assumptions for a non-virtualized operating system or application that runs in a controlled environment. However, once virtualized, the VDI may leave the controlled environment, be copied and made available to an attacker to manipulate

978-0-7695-3363-6/08 $25.00 © 2008 IEEE
DOI 10.1109/APTC.2008.17

IEEE computer society

off-line. Traditional security concepts assume a static hard-drive with limited or no physical access. Virtual disk images, however require a more dynamic and flexible security concept, similar to security requirements of mobile devices, such as laptops. Thus many non-cryptographic integrity protection mechanisms applied within the guest system may be circumvented. Similar threats apply to confidentiality protection mechanisms which, although adequate in non-virtualized systems, break down once the VDI is available off-line.
In this paper we propose our vision of a secure virtual disk image to address these security concerns.

The remainder of this paper is structured as follows: In section 2 we discuss related work followed by an introduction to the security challenges a virtual disk image faces in the context of grid computing. In section 4 we describe our prototype design of a secured disk image in detail. Section 5 outlines a typical life-cycle of a secure virtual disk image as well as non security related features. The security analysis carried out in section 6 renders an advisory model and outlines how SVDI will perform under the described circumstances. We discuss considerations which arise from the special environment and nature of secure disk images in section 7. Finally section 9 concludes and 10 discusses future work.

2 Related Work

The endeavors toward providing security in a grid environment with hostile operating systems, carried out by Mao et al. [1] show how important confidentiality and integrity are for many computing paradigms – especially in a context of grid computing. However, in contrast to the approach taken by Mao et al. we make no assumptions about the trustworthiness of OS kernels: our goal is to provide trusted storage for a wide range of virtual machines. Therefore, our approach can be regarded complementary to [1].

Terra [2] is a trusted virtual machine monitor which partitions general-purpose platforms into high assurance virtual machines. The concept of attesting to integrity for a virtual disk image has already been mentioned in this work. However, we believe by aggregating write operations into larger groups, we can reduce the overhead of cryptographic operations and thus increase performance. Moreover, our design allows a flexible integration of a snapshot capability.

OS circular is a framework for internet based virtual disk image distribution which Suzaki et al. introduced in 2007 [12]. Their primary objective is to distribute a virtual disk image to many clients. The client checks data integrity by means of a stackable virtual disk driver, based on an implementation of a trusted HTTP-FUSE CLOOP driver [12]. This approach is based on a one-to-many scheme and therefore lacks support for any updates to the VDI made by the client. Another impediment is the HTTP-FUSE CLOOP driver, which limits the OS support.

Disk encryption software in general - such as dm-crypt, TrueCrypt, FileVault and Bitlocker - are publicly available, but do not provide integrity protection. Furthermore, some solutions are tightly tailored for a specific operating system or use case. For example, Filevault is only available for Apple Macs and does not support full disk encryption. Bitlocker has support for TPM and full disk encryption but is a commercial product, only targeting Windows platforms. Hardware based disk encryption, such as Intel's Danbury, technology on the other hand, target high end corporate workstations or servers. The use of hardware or software based disk encryption products often restricts compatibility, transparency and manageability. Consequently, none of the existing products have yet targeted the flexibility and transparency needs of securing virtual disk images in a grid environment.

3 Challenges to secure images in a grid environment

In a grid environment an user is obliged to trust the resource provider in a black box model: the user submits a job and after it is finished he is presented with a result. This may lead to a conflict of interests, for example when close competitors use the same grid. It is essential in this scenario to prevent the user's data from being disclosed or altered.

Several authors [18, 19, 5], propose to utilize the concepts of Trusted Computing (TC) to enforce multilateral security and satisfy security requirements of all associated parties. By using TC combined with secure virtual disk images, the user could rely on the TC security assurance and moreover, remain in control of the data and the operating system at any time.

3.1 Threats

A traditional virtual disk image is represented by a single large file or a combination of concatenated files of a pre-defined length. The reason to use concatenated files is due to the file size restriction of legacy operating systems or file systems. As pointed out in [4] and [3], the representation of a hard disk by a loose file poses many security risks, such as theft or fraud. The threats may be broadly classified as follows:

- **Image manipulation:**
  A malicious party could alter sensitive data within the image or even inject code into a disk image without the knowledge of the legitimate owner or subsequent user.

- **Information leakage:**
  The user's credentials might be extracted from an image. Moreover a disk image may contain several snapshots, each of which may be reloaded by an attacker, providing a perfect basis for replay attacks. Most grid environments are not able to provide the fine-grained access control mechanisms consumers might want. As a result data may be exposed to competitors.

- **Image replacement:**
  No data origin authentication. An attacker may replace a file containing image data without the knowledge of the legitimate owner or subsequent user.

It is rather difficult to mitigate the threats mentioned above with traditional images while maintaining interoperability. Generally, an image may be considered at risk if moved outside the secured borders of a data center.

3.2 Design goals

To mitigate the above threats, our security goals are:

- to provide data integrity,
- to provide data confidentiality,
- to enforce the use of a specific hypervisor for a particular image.

Physical data integrity is provided by the underlying storage system such as the hard drive or network storage system. Storage systems assure that the data they receive are correct. These systems make no assumptions about the logical correctness nor freshness of the data they contain. They also lack the ability to protect their storage content from unauthorized access: often it is an all or nothing scenario in which an administrator is superior to all access control mechanisms. Thus our aim is to provide integrity and confidentiality insurance to the image owner. Additionally, we aim to preserve or extend flexibility and backward compatibility wherever possible. Moreover it is our goal to provide these security mechanisms transparently to the guest system.

Our major concern is security but we are also anxious to keep any impact on overhead and performance to a minimum.

3.3 Assumptions

We intend to make use of the concepts of trusted computing and the TPM in particular. These will be used to report the system state of a hosting environment and additionally to perform sealing and binding operations on a metafile. The guest operating system does not need necessarily to be aware of the presence of a TPM. However, the VDI implementation on the hosting system has to utilize the TPM's functions.

Also, we require the hypervisor (virtual machine monitor) and a set of userspace applications to be measured by the TPM and that the system is running in a defined state. Thus we assume that the hosting system is trusted to only execute code approved by the user. The recent work conducted by McCune et al. [6] on a reduced trusted code base outlines how a system can be trusted with trusting a minimum amount of code, whilst providing hardware-supported isolation of security-sensitive code. Past research carried out by Seshadri et al. [10] demonstrates how to ensure code integrity for commodity operating systems.

We further assume that hardware virtualization features are present on the host system to provide protected page tables as described in [10]. This becomes necessary due to a shared address space configuration in the existing Xen back-end communication system. We also assume that an integrity measurement architecture is present on the host and capable of measuring the userspace applications.

Further, we expect a trusted third party to be existent and also that trust between the hypervisor and the third party has already been established. In a grid environment a trusted third party is likely to be the resource broker. However, this requires a TPM to be present in the trusted third party as outlined in section 5.3. This also provides the basis by which trust could be established between any hosts in future grid applications. Ideally, a secure virtual disk image would fit seamlessly into an existing trusted grid infrastructure.

3.4 Use of trusted computing

As mentioned above, we propose to make use of Trusted Computing (TC) technology to protect the VDI and to utilize the services provided by a Trusted Platform Module (TPM) as defined by the Trusted Computing Group[1]. The TPM specifications [13, 14, 15], describe a tamper-resistant device with cryptographic coprocessor capabilities. This device provides the connected platform, in our case the host machine, with a number of services. These services include: special purpose registers for recording platform state; a means of reporting this state to remote entities; asymmetric key generation, encryption and digital signature capabilities. For the purposes of this paper we make use of three TC related concepts: integrity measuring, sealing and public key operations.

Integrity measuring: An integrity measurement is the cryptographic digest or hash of a platform component (i.e. a piece of software executing on the platform). For example, the integrity measurement of a program can be calcu-

[1] `http://www.trustedcomputinggroup.org`

lated by computing a cryptographic digest of a program's instruction sequence, its initial state and its input. Integrity measurements are stored in special purpose registers within the TPM called Platform Configuration Registers (PCRs).

Sealing: This is the process by which data is encrypted and associated with a set of integrity metrics representing a particular platform configuration. The protected data can only be decrypted and released for use by a TPM when the current state of the platform matches the integrity metrics to which the data was sealed.

Asymmetric keys: A TPM can generate an unlimited number of asymmetric key-pairs. For each of these pairs, private key use and mobility can be constrained, where usage is contingent upon the presence of predefined platform state (as reflected in one of the host platform's TPM PCRs). Additionally, a private key can be either migratable, non-migratable or certifiable migratable. A non-migratable key is inextricably bound to a single TPM instance, and is known only to the TPM that created it.

A certificate for a non-migratable key and its security properties may be created by the TPM on which it was generated. A certifiable migratable key (CMK) can be migrated but also retains properties which the TPM, on which the CMK was generated, can certify. When a CMK is created, control of its migration is delegated to a migration (selection) authority. In this way, controlled migration of the key is made possible, whereby an entity other than the TPM owner makes some contribution to the decision as to where the CMK can be migrated to. This ensures that the certified security properties of the key are retained.

4 Secure Disk Images

Our current design of a secure virtual disk image is built upon the existing virtual disk (block tap) driver currently integrated into Xen [17]. We base our design on the existing block tap implementation for the following reasons:

- Xen is open source and publicly available.
- An existing and well documented virtual disk driver exists.
- The existing driver offers support for portable user-level back-ends.
- Userspace tools and libraries can be used.
- Xen offers a vast range of compatibility.

The design principles, however, can easily be ported to any other virtualization technology.

Figure 1 outlines the system design of a VDI implementation in Xen.

Xen allows multiple guest operating systems to run in secured "Domains". The initial domain, Domain 0, (Dom0) is created upon first system boot. Domain 0 represents the management and control domain and thus the most privileged domains of Xen. It manages virtual devices as well as administrative tasks, such as suspension and migration of guest operating system domains. In Xen terminology, guest operating system domains are referred to as "user domains" or DomU.

Xen allows both an emulated access and a para-virtualized access to the disk interface. In a para-virtualized driver model the user domain only implements a rudimentary driver, which only forwards requests to the management domain. In the remainder of the paper we concentrate on the para-virtualized approach in favor of performance and simplicity. Consequently, modifications focus on the back-end VDI driver so all existing guest domains remain compatible. However, concepts and designs are easily applicable for an emulated access and are in no means limited to Xen itself.

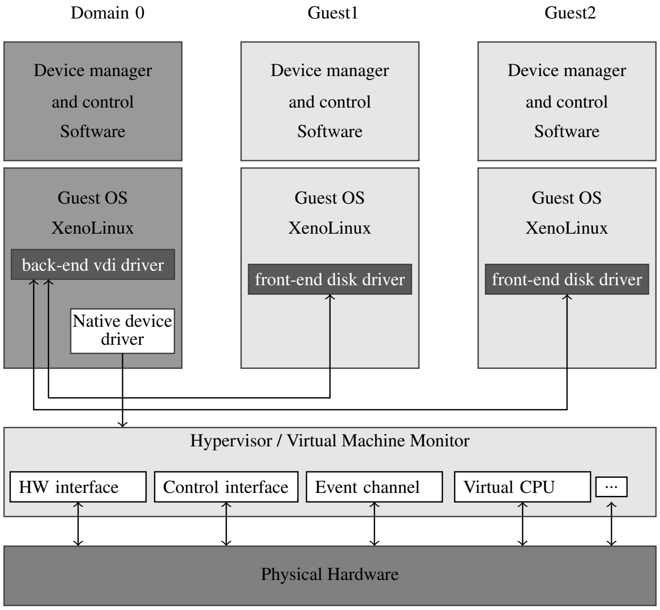

Figure 1. Sample SVDI implementation in Xen

In order to protect the integrity of a disk image, integrity metrics of the image have to be calculated, stored, and subsequently checked. As mentioned in section 3.1, a traditional VDI consists of large files. It is difficult to measure the integrity of these files in a practical and timely manner. Read-only images on the other hand can provide a checksum which has only to be checked once upon initialization. On a writable image however, every write operation performed on the image will consequently result in different integrity metrics. Timeliness therefore becomes an important consideration if the image is likely to be updated frequently during execution of applications. To avoid a constant rehashing of the virtual image we propose to split the image into smaller chunks of a fixed length. Thus only chunks that are subject to a write operation have to be measured again. Rather than placing a Merkle hash tree [8] over the image we decided to compartmentize the image into independent chunks. Therefore the integrity metrics of the chunks are independent, which allows a better parallelization of checking and generating of integrity metrics. Hence, we limit the amount of hash operation and also achieve the independence of individual chunks, which is necessary for an efficient snapshot functionality.

Our approach for a secure virtual disk image holds a set of hash values to represent the integrity metrics of the whole disk image. In this way only the areas of the image that are modified need to be re-hashed. Figure 2 illustrates how the disk image is constructed.

An alternative would be to separate read-only from writable sections of the image. However, we believe that splitting up an image into chunks has certain advantages over separating read-only from writable data on independent images. By doing so we ensure full interoperability and compatibility with existing systems as well as support for legacy operating systems that do not run on read-only images. Additionally, we gain a performance boost through the parallel processing of only those chunks that are really necessary to be updated.

In the following section we examine the detailed implementation of the proposed secure virtual disk image based on Xen.

4.1 Creating integrity metrics

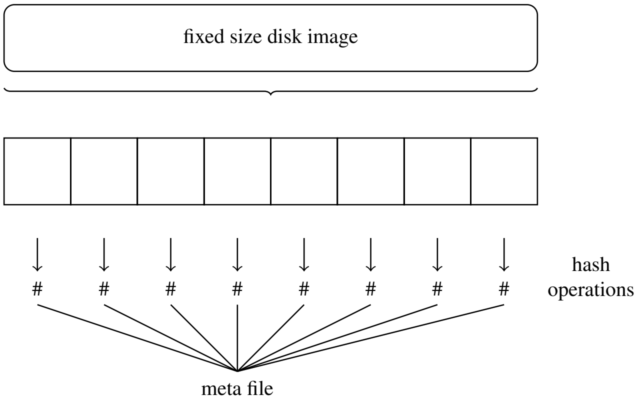

Figure 2. Overview of integrity construction

We propose to use a separate metafile to store integrity data for each chunk of the image. This could be, for example, an XML file that is associated with the VDI. Information about modification time and the corresponding chunk's hash value would be stored in this metafile. The metafile itself is encrypted, while its encryption key is protected by the sealing mechanism provided by the TPM.

A chunk is modified by a write operation of the virtualized guest. Hence the chunk is altered and, before being written to disk, hashed. Afterwards, the new hash value is then stored in the metafile.

As shown in detail in figure 3, this would be done by the front-end driver of the guest system in the user domain (DomU). This will forward a write request via a special character control device (`blktap ctrl`) to the back-end driver in Dom0. The back-end and front-end drivers in Xen communicate via an event channel and shared memory. The userspace and kernelspace part of the back-end driver communicate via named pipes and shared memory. First, the userspace control process (`tapdisk`) loads the metafile, by invoking the `block svdi` module. This module then makes a syscall to the *libTPM* to unseal the metafile. After being unsealed the metafile is stored and updated in the host's memory until the guest system is finally paused or halted. The userspace control process will then invoke *libcrypto* to generate a hash value for the chunk. Finally this process will invoke a system call via *libaio*[2] to write the chunk to disk. At the same time the control process updates the hash code contained in the metafile.

This allows transparent and parallel operations for multiple virtual machines. Implementation overhead is very low and limited to the two objects we propose in this paper - the `block svdi` module as well as to the `trust ctrl` process. User space libraries such as *libcrypto* and *libaio* are widely available, *libTPM* is available, for instance, via the TrouSerS software project[3]. We are aware that this will result in additional read/write operations that will impact performance but we consider this a fair compromise for security.

4.2 Metafile

We require the metafile to be confidential and tamper-evident, as an attacker might alter any chunks and the corresponding hash codes are stored in the metafile. The metafile will also store the encryption key for the chunks and thus needs special attention. The file itself may be protected by encrypting and signing its content when the virtual machine

[2] asynchronous input/output library

[3] `http://trousers.sourceforge.net/`

is suspended or powered down. This metafile is bundled together with the data chunks and may also reside on untrusted storage. The metafile should, however, be protected in a manner that its content can only be revealed if the host system is in a pre-defined state. Thus the metafile should be sealed to a specific TPM and only be unsealed if the system is in a pre-defined state. As a consequence the metafile has to be maintained by the same object which manages the TPM hardware - in this case Domain0.
Upon the first read request to the disk image, the userspace control program calls *libTPM* to unseal the metafile, hence gaining the encryption credential for the metafile. Afterwards, the userspace control program uses *libcrypto* to ensure the hash codes contained in the metafile are correct, if so, the control program will process the guest's request. Once loaded, the metafile resides in memory and is constantly being updated throughout runtime. If the guest system is powered down or suspended, the control program measures and stores the metafile's new hash code, encrypts it, and seals the metafile's encryption key. Listing 1 outlines a sample metafile.

In favor of a snapshot capability 5.7, we decided against the obvious approach of mapping block addresses to chunk names. We propose to implement an unique random number (`NextFreeChunk`) as an addition to a basename (e.g. chunk.[i]). By doing so, no information about the chunk's source or allocation is revealed. The metafile consequently reflects the mapping of block address and file name through the `BlockAddress` directive.

```
<sampleImage>
 <header>
  ...
   <SHA256>894f435gd ... fas32dag</SHA256>
   <EncryptionKey>3b23894f ... fce3bc95</EncryptionKey>
   <EncryptionAlgorithm>AES</EncryptionAlgorithm>
   <ChunkSize>16777216</ChunkSize>
   <ImageSize>536870912000</ImageSize>
   <NextFreeChunk>123</NextFreeChunk>
   <SnapshotVersion>2</SnapshotVersion>
 </header>
  ...
 <chunk.122>
   <SnapshotVersion>2</SnapshotVersion>
   <BlockAddress>00040000</BlockAddress>
   <ChunkPath>/sampleImage/chunk.122</ChunkPath>
   <SHA256>dc460da4ad72c...6899d54ef98b5</SHA256>
  ...
 </chunk.122>
</sampleImage>
```

Listing 1. Sample metafile

We refer to the chunk size via the `ChunkSize` element in the metafile. A small chunk size would result in a constant, but unnecessary re-hash if the block has changed. A large chunk size on the other hand would result in an increased runtime. To reduce additional read/write overhead and the amount of hash operations, we propose a fixed chunk size equal to the cache size of the underlying hard-drive. By choosing chunk sizes of 8 to 16 MegaBytes the performance impact of multiple chunk read/write operations, for instance caused by fragmentation, can be further reduced.

4.3 Checking Integrity

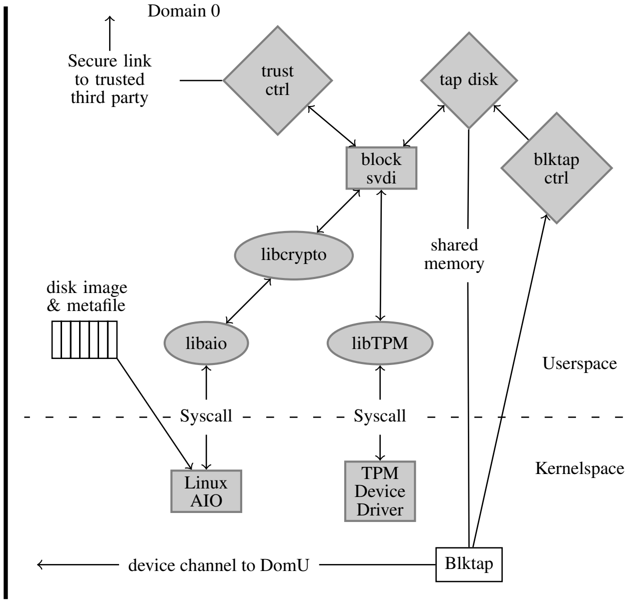

Figure 3. Sample SVDI implementation in Xen

The integrity checks can be performed in two ways:

4.3.1 Checking the integrity during operation

When the virtualized guest requires access to data, the appropiate chunk of the VDI gets loaded, hashed, and the result is compared to the value stored in the metafile. Analogous to the write operation described in section 4.1, a read request will be forwarded from the guest system's front-end driver to the Domain 0 back-end driver via the control device. In response to the first read request the userspace control program unseals and loads the metafile by calling *libTPM*. Next, the control program invokes *libaio* to read the chunk, and *libcrypto* to perform the hash operation. This is followed by a comparison of the result and the pre-stored hash value. Assuming an enforcing policy, the back-end implementation will prevent further execution if the hash values do not match - e.g. by pausing the affected virtual machine. Alternatively, a reporting only policy could be applied allowing the operation to proceed but reporting the incident. If the guest system needs to be informed about an event, modification to the guest system's disk driver or the virtual support application have to be implemented. Since the metafile is sealed to a specific set of host platform integrity metrics these assumptions are not unreasonable. The metafile would only be unsealed if the host was running a process to enforce the above policies.

4.3.2 Checking the integrity before operation

Before powering up, resuming, or migration to another host, the integrity of each chunk is checked against its pre-stored value. In a manner analogous to that above, an enforcing or reporting only policy may be applied. The result of each hashing operation is independent from the previous one and thus hashing can be performed in parallel. Compared to a traditional approach of hashing a single file our approach offers the speedup of a parallelized algorithm. Moreover, it is not necessary to measure the complete image before operation and, even if one chunk fails to validate, the remainder of the image may still be considered uncorrupted.

4.4 Ensuring confidentiality

Confidentiality for images is provided by encrypting each chunk of the image with a suitable encryption algorithm. The key for the image may be held in the metafile, or alternatively, the user may input a key upon powering up or migrating the virtual disk image. Xen already implements the QEMU[4] disk image format (qcow), which integrates 128 bit AES encryption [7]. The qcow disk image format is a versatile Copy-on-Write image format for the open source virtual machine monitor QEMU. However, in favor of flexibility we propose the use of *libcrypto* which offers a different set of crypto algorithms.

Since the metafile itself is sealed to a TPM and a certain system state, only one particular system in a particular state may reveal any encryption key stored in the file. Thus the virtual disk image can securely reside on an untrusted storage location - e.g. a storage provider located on the Internet. The existing storage infrastructure can be maintained or enhanced by any untrusted storage provider without compromising security.

5 Characteristics

Aside from the security attributes described in section 4, a worthwhile operation requires the SVDI to fulfil an additional set of features. In the following section we describe how SVDI meets the requirements of a modern virtualized infrastructure and discuss the typical life-cycle of a secure virtual disk image.

5.1 Initialization

Creating a new SVDI only requires the initialization of one metafile per disk image instance. Disk images can grow dynamically during their lifetime due to their sparse capability described in 5.4. It is still necessary, however, to specify its maximum capacity upon creation as it is treated by the guest as a hard-drive of a fixed length. Other parameters, such as encryption or hash algorithms, may be specified during generation or set to a default value. After triggering the initialization procedure a random encryption key is generated and stored in the metafile. Depending on the usage scenario, the initialization procedure itself can take place on the host, at a trusted third party, or outside the datacenter.

Finally, the metafile is sealed to its destination host using the trusted computing sealing mechanism. The destination host is then able to use the metafile and proceed with normal operation. Each chunk will be created dynamically by the back-end driver if a virtualized guest writes to the corresponding area of the image.

5.2 Deletion

Our main design goals are to maintain confidentiality, integrity and authentication, throughout the disk image's life-cycle. Consequently, those characteristics should still be intact once the disk image reaches the end of its lifetime. Moreover, the weakest attribute for confidentiality is the encryption key and its backups. The metafile is the digital equivalent to a key and thus it needs to be kept secure throughout the disk image life-cycle. If the metafile and the encryption key are securely deleted then this effectively deletes the image. This requires careful control over the distribution of the metafile and encryption key. Ensuring that the metafile can only be unsealed on a particular platform, gives some degree of control on the file's distribution.

5.3 Backup

An image is bound to a specific platform during operation. Recovering data from this platform is difficult if it fails while the image is bound to it. As a consequence we may allow a trusted third party or the disk image provider to keep a backup of the metafile and/or encryption key. The metafile must never be stored in the clear, hence we require the trusted third party to protect the metafile copy by sealing it to its own TPM. However, if a host fails during operation and the metafile needs to be recovered, integrity metrics may not be up-to-date. Transferring a metafile backup to a new host takes place as outlined in section (5.6).

[4] http://fabrice.bellard.free.fr/qemu/

5.4 Sparse format

Secure virtual disk images utilize disk space more efficiently by saving disk content in a sparse manner. Instead of writing out allocated, but empty disk space, secure VDIs only store abbreviated information about those areas. Primarily it allows data to be saved more efficiently and at the same time allow images to grow during operation.

5.5 Garbage collection

The way we designed the disk images makes file deletion a challenging task: Once an address space is allocated and eventually released (e.g file deletion), the underlying disk image implementation does not reveal that free space automatically. Therefore a garbage collection instance becomes necessary. Without modifying the overlying guest OS or keeping a block address mapping log, garbage collection is far from trivial. Our current approach to reclaim free space anticipates an offline garbage collection mechanism. Thus, an image is scanned for free space while the virtual machine is powered off and an exclusive access right is granted. This offers three major advantages:

- no guest OS modifications
- no additional performance overhead
- the possibility to incorporate defragmentation.

However, further work is required in the future on this issue to present an online garbage collection implementation.

5.6 Migration

5.6.1 Offline migration

The possibility to migrate a virtual machine in a secure and flexible way to another physical host is vital in the context of grid computing. The secure VDI we presented earlier allows an easy and secure offline migration - a migration while the guest is powered down.

One of our previous assumptions in section 3.3 requires the hosting machine to be up and in a trustworthy state. This allows us to easily migrate the protection key of the metafile. We utilize a trusted control process (`trust ctrl`), to interact with the trusted resource broker and the back-end (`block svdi`), implementation. The trusted third party triggers and manages the migration process. Once the chunk files are accessible by the targeted machine the metafile is migrated to the new host. In detail, the migration process of the metafile takes place as follows: If the guest system is running it is powered off by the `trust ctrl` application. At this stage the userspace control program stores the remaining data on disk then updates and seals the metafile as described in section 4.1. In order to prevent replay attacks on the disk image the `trust ctrl` application will only allow migration if the virtual machine is not running. This is followed by invoking the *libTPM* to call “TPM_MigrateKey”, and temporarily migrate the key to the trusted resource broker. The trusted resource broker then decides which machine the key will be migrated to. Once the key is migrated onto the target platform’s TPM, the destination `trust ctrl` application can unwrap the metafile. Hence the disk image can be decrypted, checked, and finally the guest machine can be powered up. In a suspended state memory content could also be encrypted using an encryption key provided in the metafile.

5.6.2 Live migration

By utilizing the trusted computing mechanisms the metafile is sealed to the new host’s TPM and its system state. This way we ensure a tamper-free migration whilst at the same time maintain full flexibility and compatibility. As stressed by Oberheide et al. [9], live migration of virtual machines poses a security issue. Oberheide pointed out that a migrating virtual machine can be manipulated whilst traveling across an unsecured network. We believe by utilizing the concepts of trusted computing as well as a trusted third party (e.g. resource broker), a secure link between migrating hosts can be established and therefore perform a secure migration. Secure live migration is, however, beyond the scope of this paper.

5.7 Snapshots

We decided to implement block address mapping into the metafile to allow a snapshot feature. In a snapshot-less scenario a typical modification to a chunk would take place as outlined in section 4.1. First read, update and hash the chunk, followed by writing it to disk.

A snapshot however, is triggered by increasing a `SnapshotVersion` in the metafile’s header section. After a snapshot request is made the back-end driver treats every chunk as read-only. Then, in a copy-on-write manner, a write operation causes the driver to read the chunk, perform the requested write, calculate the integrity metrics, and finally allocate a new sequence number for this chunk. In the metafile this will be reflected by a `SnapshotVersion` directive in the appropriate section. In addition, the metafile and the chunks currently in operation are written to the storage location in order to ensure a crash consistent state. This allows snapshots to be taken whilst the disk is in use or, alternatively, if the virtual machine is suspended or halted. Chunks which do not contain a snapshot version are valid for all snapshot revisions.

We consider snapshotting as a valuable and desirable feature and decided to provide this component, even though it results in more fragmentation. By using a fixed chunk size a snapshot will create fragments and waste a certain amount of storage space.

6 Security Analysis

6.1 Threat model

With our adversary model we target the unique threats which an image in a grid environment may be exposed to. In our threat model we assume an adversary who has access to the storage location of a virtual disk image. This attacker may read, modify, and inject data into the image file. Additionally, we presume an adversary can read, modify, inject, and replay any arbitrary information exchanged between the disk image storage location and the host system. As with the trusted computing model, we assume that the attacker does not have physical access to the host platform. Moreover, we assume that the trusted third party is trustworthy.

6.2 Analysis

In the previous section, we outlined how we achieve our initial security goals to provide data secrecy, consistency and integrity for disk images. We further demonstrate in the following, how SVDIs will perform in response to the described attack scenarios:

6.2.1 Confidentiality attack

Attempting to gain access to information without authorization. The encryption key for the SVDI is held within the metafile. Thus the confidentiality of the metafile is important. The metafile itself is protected by the trusted computing sealing mechanism. Thus to break confidentiality, an attacker would have to compromise the trusted system or gain access to the TPM's private storage root key. However, for backup or migration purpose, we allow a copy of the metafile to be stored by the trusted third party. Here, the metafile is again sealed by the TPM on the trusted third party's platform. Hence, if the metafile cannot be revealed outside a trusted environment, confidentiality remains intact.

6.2.2 Integrity attack

Modifying data without authorization. Legitimate updates of hash codes in the metafile can only be made if the system provides the access credential outlined earlier. Any unauthorized modification to chunks or to metadata will be uncovered by a mismatched checksum. Rolling back to a previous version of a chunk will have the right encryption key but the wrong integrity metrics and, therefore, can be easily detected. Additionally, swapping the order of the chunks will also be identified by inconsistent checksums. The metafile itself cannot be altered while in storage or transit, as it is cryptographically protected. It carrries a signed integrity metrics of itself. However, as long as the trusted computing framework remains intact, an attacker will fail to inject malicious integrity metrics into the metafile.

6.2.3 Authentication attack

A man-in-the middle attack is ineffective because no unencrypted data ever leaves the host or guest system nor the trusted third party. An adversary may, however, present a malicious metafile to the host. In this case, the host will be unable to unseal the metafile as the host expects the metafile to be sealed by its own TPM. The SVDI implementation will therefore reject all non-valid metafiles.

6.2.4 Availability attack

An adversary who has control over the storage location or network may change, append data, or delete chunks and/or metadata. Modifications will be easily detected, however this may render the guest machine un-operational. Nevertheless, if an attacker has high privileges he might also shut the network or the storage location down. Alternatively, an attacker may fake latency to disrupt or defer a normal operation. Such general denial of service attacks are beyond the scope of our stated security goals. A more enhanced version of secure virtual disk images may make redundant chunks available to realize multiplexed storage - similar to a raid array. This could make secure virtual disk images resistant to availability attacks but inflicting an additional write overhead.

7 Considerations

While our design offers many beneficial security attributes it raises different concerns that need to be addressed for every day usage. Consequently, the following section discusses several usage scenarios which need special attention.

7.1 Fragmentation

Fragmentation poses a performance and space issue for secure VDIs. SVDIs provide a hard-drive implementation and therefore file handling is managed by the overlying file system structure. Different file systems implement various mechanisms to prevent fragmentation, however, the file system is not aware how the disk implementation is handled by the SVDI. Any file could be spread across multiple chunks and invoke multiple hash and crypto operations if it is altered. Thus, too many fragments located throughout the disk-layout could inflict a performance overhead. The use of file systems which group blocks such as UFS (Unix File System), or FFS (Fast File System), promise to mitigate the fragmentation issue in this case.

Snapshots, on the other hand, will create fragmentation since new chunks with redundant information are created for every snapshot version.

7.2 Performance

This approach has been designed with security, rather then performance, in mind. Constantly performing hash operations certainly will have an impact on performance. It will result in slightly higher disk I/O operations and in an increased memory and CPU usage - due to hash and crypto operations. Nevertheless, by spreading chunks across multiple storage locations a raid array like feature could be achieved without inflicting any overhead. Depending on the security requirements of an application, the security gained could be considered more important than the performance impact.

7.3 Swap-space

Files that might change frequently, such as swap-space or temporary files, would cause a constant re-hashing even though their integrity might not be of importance. To reduce the performance impact, those files could be sourced out into separate, encryption only VDIs. For swap-space for instance, this could be done by creating an encrypted disk image with a random encryption key each time the virtual machine is powered up. Alternatively, encryption keys for swap-space could be included into the metafile.

8 Usage scenario

Secure virtual disk images offer new possibilities in grid usage scenarios. For instance, a grid customer may provide his own disk image including his own operating system, application, and data to the grid without ever exposing the content of the disk to anyone other than a specific grid node. Moreover, the grid customer is given the ability to retain control of his image throughout its entire lifetime. By utilizing the concepts of trusted computing the owner can additionally ensure that only a particular platform, in a well defined state, can process this image. This enables a grid consumer to gain a new level of data security assurance.

9 Conclusion

With the approach described in this paper we are able to deliver confidentiality and integrity to virtual disk images. Our design is based on the foundation of trusted computing and virtualization itself. We believe that protecting virtual disk images, not only in the context of grid computing but also in general, is important for the future evolution of virtualization. Within a data center, security mechanisms in hardware or software may be utilized to protect a disk image but, as described in [4] and [3], it is important to protect an image if it is moved outside the secured borders of the data center. Further, we are able to provide those attributes while the underlaying storage cannot guarantee any of these. Data on virtualized disks is protected as a whole including log files, application, as well as any other part of the operating system. With SVDIs we are able to transparently apply confidentiality and integrity to any commodity or legacy operating system.

Secure virtual disk images allow location and storage transparent way to provide security to a virtualized guest, hence, an image could reside on any untrusted storage.

10 Future Work

In this paper we demonstrated the principles and goals behind secure virtual disk images. Future work will incorporate a more detailed proof-of-concept implementation and will reveal where work remains to be done. For instance, garbage collection and fragmentation are not yet solved in a completely satisfactory manner. Modifications to the guest systems front-end disk driver promise to mitigate both issues but require modifications to the guest system, which will also require maintenance work in the future. Further, a proof-of-concept implementation has to prove its capabilities in a thorough performance analysis.

As already pointed out in 5.6, live migration poses a difficult design challenge. Not only the system's state but also its current memory content has to be transferred to a different host in a fast, but secure, manner. This issue remains the subject of our future work.

Our future for virtual disk images is to unify all different virtual disk image approaches to one secure virtual disk image standard. The Open Virtual Machine Format (OVF) proposed by VMware and XenSource in [16], targets an open and neutral standard for virtual appliances. Incorporating our secure virtual disk images into the OVF could deliver confidentiality and integrity assurance to those. A general and vendor neutral secure disk image standard could be beneficial for a vast range of future virtual applications.

11 Acknowledgments

Discussion and feedback from Steffen Reidt and Frederic Stumpf were very valuable, as were the comments of our anonymous reviewers.

References

[1] H. Chen, J. Chen, W. Mao, and F. Yan. Daonity - grid security from two levels of virtualization. *Inf. Secur. Tech. Rep.*, 12(3):123–138, 2007.

[2] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 193–206, New York, NY, USA, 2003. ACM.

[3] T. Garfinkel and M. Rosenblum. When virtual is harder than real: security challenges in virtual machine based computing environments. In *HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems*, pages 20–20, Berkeley, CA, USA, 2005. USENIX Association.

[4] C. Gebhardt and A. Tomlinson. Security considerations for virtualization. Technical report, Department of Mathematics, Royal Holloway, University of London, 2008.

[5] H. Lohr, H. V. Ramasamy, A.-R. Sadeghi, S. Schulz, M. Schunter, and C. Stuble. Enhancing grid security using trusted virtualization. In B. Xiao, L. T. Yang, J. Ma, C. Muller-Schloer, and Y. Hua, editors, *ATC*, volume 4610 of *Lecture Notes in Computer Science*, pages 372–384. Springer, 2007.

[6] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: an execution infrastructure for tcb minimization. *SIGOPS Oper. Syst. Rev.*, 42(4):315–328, 2008.

[7] M. McLoughlin. The qcow image format. `http://www.gnome.org/~markmc/qcow-image-format.html`.

[8] R. C. Merkle. Protocols for public key cryptosystems. *Security and Privacy*, 00:122–134, 1980.

[9] J. Oberheide, E. Cooke, and F. Jahanian. Exploiting live virtual machine migration. Black Hat DC, Washington DC, February 2008.

[10] A. Seshadri, M. Luk, N. Qu, and A. Perrig. Secvisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 335–350, New York, NY, USA, 2007. ACM.

[11] C. Strachey. Time sharing in large fast computers. volume paper B. 2. 1, pages 336–341. Proceedings of the International Conference on Information Processing, June 1959.

[12] K. Suzaki, T. Yagi, K. Iijima, and N. A. Quynh. Os circular: internet client for reference. In *LISA'07: Proceedings of the 21st conference on 21st Large Installation System Administration Conference*, pages 1–12, Berkeley, CA, USA, 2007. USENIX Association.

[13] TCG. TPM Main, Part 1 Design Principles. TCG Specification Version 1.2 Revision 103, The Trusted Computing Group, Portland, OR, USA, July 2007.

[14] TCG. TPM Main, Part 2 TPM Data Structures. TCG Specification Version 1.2 Revision 103, The Trusted Computing Group, Portland, OR, USA, July 2007.

[15] TCG. TPM Main, Part 3 Commands. TCG Specification Version 1.2 Revision 103, The Trusted Computing Group, Portland, OR, USA, July 2007.

[16] VMware and XenSource. The open virtual machine format whitepaper for ovf specification. Technical report, VMware and XenSource, 2007.

[17] A. Warfield and J. Chesterfield. Blktap userspace tools + library. `http://lxr.xensource.com/lxr/source/tools/blktap/README`, June 2006.

[18] Wenbo Mao, Andrew Martin, Hai Jin, and Huanguo Zhang. Innovations for grid security from trusted computing. In *Fourteenth International Workshop on Security Protocols*, LNCS. Springer-Verlag, 2006.

[19] P.-W. Yau, A. Tomlinson, S. Balfe, and E. Gallery. Securing grid workflows with trusted computing. In *8th IEEE International Symposium on Cluster Computing and the Grid*, May 2008.