Trusted Computing and Secure Virtualization in Cloud Computing

Master Thesis

Nicolae Paladi

Luleå University of Technology
Dept. of Computer Science,
Electrical and Space Engineering
Div. of Computer and Systems Science

Swedish Institute of Computer Science
Secure Systems Group

August 7, 2012

ABSTRACT

Large-scale deployment and use of cloud computing in industry is accompanied and in the same time hampered by concerns regarding protection of data handled by cloud computing providers. One of the consequences of moving data processing and storage off company premises is that organizations have less control over their infrastructure. As a result, cloud service (CS) clients must trust that the CS provider is able to protect their data and infrastructure from both external and internal attacks. Currently however, such trust can only rely on organizational processes declared by the CS provider and can not be remotely verified and validated by an external party.

Enabling the CS client to verify the integrity of the host where the virtual machine instance will run, as well as to ensure that the virtual machine image has not been tampered with, are some steps towards building trust in the CS provider. Having the tools to perform such verifications prior to the launch of the VM instance allows the CS clients to decide in runtime whether certain data should be stored- or calculations should be made on the VM instance offered by the CS provider.

This thesis combines three components – trusted computing, virtualization technology and cloud computing platforms – to address issues of trust and security in public cloud computing environments. Of the three components, virtualization technology has had the longest evolution and is a cornerstone for the realization of cloud computing. Trusted computing is a recent industry initiative that aims to implement the root of trust in a hardware component, the trusted platform module. The initiative has been formalized in a set of specifications and is currently at version 1.2. Cloud computing platforms pool virtualized computing, storage and network resources in order to serve a large number of customers customers that use a multi-tenant multiplexing model to offer on-demand self-service over broad network. Open source cloud computing platforms are, similar to trusted computing, a fairly recent technology in active development.

The issue of trust in public cloud environments is addressed by examining the state of the art within cloud computing security and subsequently addressing the issues of establishing trust in the launch of a generic virtual machine in a public cloud environment. As a result, the thesis proposes a trusted launch protocol that allows CS clients to verify and ensure the integrity of the VM instance at launch time, as well as the integrity of the host where the VM instance is launched. The protocol relies on the use of Trusted Platform Module (TPM) for key generation and data protection. The TPM also plays an essential part in the integrity attestation of the VM instance host. Along with a theoretical, platform-agnostic protocol, the thesis also describes a detailed implementation design of the protocol using the OpenStack cloud computing platform.

In order the verify the implementability of the proposed protocol, a prototype implementation has built using a distributed deployment of OpenStack.

While the protocol covers only the trusted launch procedure using generic virtual machine images, it presents a step aimed to contribute towards the creation of a secure and trusted public cloud computing environment.

ACKNOWLEDGEMENTS

This thesis has been carried out at the Swedish Institute of Computer Science (SICS) in Stockholm, Kista under the supervision of Dr. Christian Gehrmann. The thesis was part of the trusted and secure virtualization environments project, a joint research project between SICS and Ericsson Research. I would like to thank Mudassar Aslam (SICS) for his input and help in understanding the essence of trusted computing and Fredric Morenius (Ericsson Research) for his comments and help throughout the protocol design and implementation phase. Especially helpful were Lars Rasmusson's (SICS) tips on the use of Python and the dive into the Linux kernel code investigating LINUX-IMA internals.

I would also like to thank Professor Tero Päivärinta from Luleå University of Technology (LTU) for his comments and suggestions throughout the thesis, as they have considerably helped shape and structure this work, especially when it comes to methodological aspects. Similarly, I would like to thank Todd Booth (LTU) for offering helpful feedback on the launch protocol.

Special thanks to Alex Averbuch (SICS) for his feedback, constructive criticism and help in brainstorming solutions to seemingly unsurmountable issues.

Last but not least, I would express my gratitude to my family, girlfriend, and close friends for their unyielding support, tolerance and curiosity in the topic.

It's been an epic journey into a year of changes.

Nicolae Paladi

List of Abbreviations

- *AIK* – Attestation Identity Key
- *CTRM* – Core Root of Trust Management
- *EK* – Endorsement Key
- *IaaS* – Infrastructure as a Service
- *IMA* – Integrity Measurement Architecture
- *GRUB* – GRand Unified Bootloader
- *GVMI* – Generic Virtual Machine Image
- *PCR* – Platform Configuration Registry
- *PK* – Public Key
- *PrK* – Private Key
- *RNG* – Random Number Generator
- *TCB* – Trusted Computing Base
- *TCG* – Trusted Computing Group
- *TPM* – Trusted Platform Module
- *TTP* – Trusted Third Party
- *TSS* – TCG Software Stack
- *SLA* – Service Level Agreement
- *SRK* – Storage Root Key
- *VM* – Virtual Machine
- *VMI* – Virtual Machine Image

Contents

Chapter 1
Introduction

1.1 The Promise of Cloud Computing

In spite of the rapid expansion of Infrastructure-as-a-Service (IaaS) technologies such as Amazon EC2 [1], Microsoft Azure [2], services provided by RackSpace [3] and others, IaaS services continue to be plagued by vulnerabilities at several levels of the software stack, from the web based cloud management console [1] to VM side-channel attacks, to information leakage, to collocated malicious virtual machine instances [2]. The need for secure cloud storage and cloud computing environments has been reiterated on numerous occasions. For example, Molnar et al [3] cite industry decision makers to emphasize the fact that security concerns are among the major factors that prevent businesses from deploying their data and computations into the cloud. Common reasons are unawareness of the state of the data and algorithms once it is in the cloud environment, as well as concerns regarding cloud provider bankruptcy and subsequent lack of clarity and established procedures of data protection and retrieval, along with many other examples. Similarly, Chen et al [4] cite opinions originating from academia, government and industry that point to security concerns as a barrier preventing a quicker adoption of cloud computing. The reasons are both technical, such as the fear of data loss, data breach and data tampering as well as organizational, such as reputation fatesharing. Similar views are reported by other researchers within cloud computing security ([5, 6]).

The economic benefits of using cloud storage and cloud computing are appealing enough to promote adoption of these technologies, hence their use is likely to increase over time [4]. In this situation, there is a risk that the economic benefits obtained today through the rapid adoption of cloud technologies will in some cases be compensated or even overcompensated by losses resulting from unexpected lack of availability as well as theft and corruption of data.

The continuous flow of vulnerabilities discovered in the software stack underlying IaaS platforms has prompted the move towards implementing trust anchors into hardware. Although this move has the potential to greatly reduce the risks posed by software vulnerabilities, it does not guarantee a secure platform out of the box. Rather, the results depend on the correct usage of the trusted hardware.

The Trusted Computing initiative and adoption of *trusted platform modules* (TPM) has been steadily gaining momentum since it's inception [7]. Participation of hardware manufacturing industry leaders in the Trusted Computing Group [4] is likely to accelerate the adoption of this technology across hardware architectures and platforms. Following its initial predominance and narrow focus on laptop computers,

[1]Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/
[2]Microsoft Azure, http://www.microsoft.com/windowsazure/
[3]Rackspace Cloud Hosting, http://www.rackspace.com/cloud/cloud_hosting_products/servers/
[4]Trusted computing group website, http://www.trustedcomputinggroup.org/about_tcg

trusted computing is making its way into new devices. For example, the use of trusted computing on mobile platforms is already the focus of several recent research projects [8, 9] with more to come as increased functionality and ever more information stored on mobile devices become more attractive targets for malware.

Another important application domain of trusted computing is its use in virtualized systems and cloud computing [10]. Trustworthy integrity verification of the software components used within the cloud computing infrastructure, as well as information protection using trusted computing techniques can address some of the security concerns related to off-premises computing. While it does not actually offer absolute guarantees, trusted computing raises the complexity bar for attackers by placing the root of trust at the hardware level [5]. With a correct implementation, an attacker would need physical access to the hardware in order to subvert the TPM [11]. However, as the technology is still new and in active development, the best practices for the use of TPM are yet to be identified. This is especially relevant for virtualized environments and trusted cloud computing, where the functionality of a single TPM chip needs to be shared between several virtual machines. Solutions like virtualization of TPMs [12] create new possibilities for implementation of secure launch and secure migration of VMs [13, 14]. In the same time new attack techniques demonstrate that software implementation of TPM increases the trusted computing base (TCB) and introduces new vulnerabilities [15]. This implies that new solutions for secure VM launch and migration need to be found based on the existing components of the TPM and with minimal changes to the TCB.

1.2 Problem Outline

The four message protection classes available in the current specification of the TPM (*binding, signing, sealing and signed sealing*), together with the encryption and signature keys available to the TPM (further described in chapter 2) provide a powerful set of tools that can be used for trusted launch and migration of VMs in cloud environments. As an example, based on some of these tools Santos proposed a secure launch and migration protocol which relies on a third-party *trusted coordinator* to attest the TPM-enabled nodes and uses the capabilities of the hardware TPM chip [5]. Other researchers have proposed a set of migration protocols that rely on TPM virtualization ([13, 14]).

This paper describes a secure VM launch protocol that can be implemented in one of the existing open source cloud operating systems. The solution has been guided by the following requirements:

- R1: *The launch should be trustable, so that a user has the mechanisms to ensure that the VM has been launch or migrated to a trustworthy host.*
- R2: *the client should have the possibility to reliably determine that it is communicating the the generic VM launched on a secure host, and not with a different generic VM instance.*
- R3: *The integrity of the VM must be verifiable by the target node.*
- R4: *The trusted VM launch procedure should be scalable and have a minimum impact on the performance of the cloud computing platform.*
- R5: *Users should have a transparent view of the secure launch procedures.*

The protocol makes use of TPM protection classes and available signature and encryption keys to ensure a secure VM launch procedure on cloud computing platforms.

[5]Trusted Platform Module main specification http://www.trustedcomputinggroup.org/resources/tpm_main_specification

1.3 Thesis outline

Chapter 2 presents an overview of cloud computing, trusted computing and virtualization, we well as a review of the security concerns related to the current cloud computing model and continues with an overview of the state of the art in cloud security, focusing on threat models, exploits and attack techniques jeopardizing security of public cloud computing. Chapter 3 formulates the scope of the problem examined throughout this thesis and defines two research propositions. Chapter 4 contains a review of the research approach employed throughout this study. Chapter 5 contains the theoretical contribution of the study, which addresses the issues described in the defined propositions. Chapter 6 contains a detailed description of the implementation of the solution formulated in the theoretical contribution of the study as well as a discussion of the implementation results. The thesis concludes with a set of protocol implementation recommendations and further research suggestions in chapter 7.

Chapter 2

Security Aspects of Cloud Computing and Trusted Computing

The term cloud computing, which is associated with the new paradigm for provisioning of computing infrastructure is still poorly defined and understood, and is often interpreted as a reincarnation of grid computing [16].

Provisioning of computational resources over the network has been available as a tool at different scales, ranging from `distcc`, [1] used between several user-owned computational devices, to the ambitious MilklyWay@Home project [2], which harnesses the unused computational power of personal PCs in order to calculate a 3-dimensional map of the Milky Way galaxy.

However, the current definition of cloud computing focuses on a centralized provisioning of computational resources to multiple remote clients. Based on a review of 21 publications, Vaquero et al proposed in [16] the following definition of cloud computing:

> Clouds are a large pools of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs.

This paradigm became popularized among businesses as a way to reduce upfront infrastructure investments, maintenance costs and eventual replacement costs. After a brief introduction to the structure of cloud computing, we will focus on the risks related to cloud computing and the building blocks of its security model. While the definition provided by Vaquero et al offers broad perspective of cloud computing, other definitions will be used throughout the study in order to emphasize specific aspects, such as security risks or infrastructure architecture.

2.1 Cloud Computing Basics

Along the lines of the above definition, cloud computing offers *on-demand self-service* over *broad network access* by employing *resource pooling* in order to serve multiple customers using a multi-tenant model

[1] DistCC: a free, distributed C/C++ compiler `http://code.google.com/p/distcc/`

[2] MilkyWay@Home `http://spacehack.org/project/milkywayhome`

[17]. In this case, the physical location of the data is independent from its representation, so the users have no control nor knowledge of the physical placement of the data. Important capabilities of cloud computing are its *rapid elasticity* that allows to scale the provided computational and storage resources in line with the demand, as well as the built-in capability to *measure the service* at an appropriate level of abstraction (e.g. storage, processor time, bandwidth, active user accounts, etc.). Such an approach to measuring the service provides transparent picture of the utilized service to both the user and the provider of the service [17].

2.1.1 Service Classification

Two other aspects that are important for the understanding of the cloud computing paradigm are its *service models* and *deployment models*.

There are three widely adopted service models for cloud computing:

- *Software as a Service (SaaS)* – in this model, the user has the capability to use the provider's applications which are deployed on a cloud infrastructure. In this case, all of the underlying implementation and deployment is normally abstracted from the user and only a limited set of configuration controls are made available. Similarly, data created by the SaaS applications is transparently stored in the cloud infrastructure.

- *Platform as a Service (PaaS)* – allows a wider range of capabilities for the user, providing the ability to deploy onto the cloud infrastructure applications created and acquired by the user, within the frame of the development languages, application programming interfaces (APIs) and services that are made available by the provider. The user has broad control of the deployed applications and data, however does not have control of the underlying computing infrastructure.

- *Infrastructure as a Service* – allows the user to provision processing power, disk storage, random access memory, network capabilities et cetera. The user can use the allocated resources in order to develop, deploy and run arbitrary software using the provisioned computational resources. In this case, the user is still using a sandboxed environment, where they have broad control over the provisioned resources, but no control over the underlying cloud management infrastructure. This thesis focuses on certain aspects of the IaaS with regard to security and trustworthiness of the provisioned computational resources with respect to both third parties and the IaaS provider itself.

There are four generic types of cloud deployment models: *private clouds*, *public clouds*, *community clouds* and *hybrid clouds*. NIST [17] provides more details about the characteristics of the of the models. In the context of the current thesis focusing on trusted computing in cloud environments, we are mostly interested in the distinction between *private clouds* and *other types of clouds*. In the former case, the full stack forming the cloud deployment is part of the customer's security perimeter and the customer has potentially full control over the hardware, network and software components. In the latter case, the cloud deployment infrastructure is either partially or fully placed on the premises of other organizations, hence limiting the capabilities of the client to monitor and control the infrastructure. This thesis focuses on aspects of trusted computing in clouds of the second type, collectively denoted as *public clouds*.

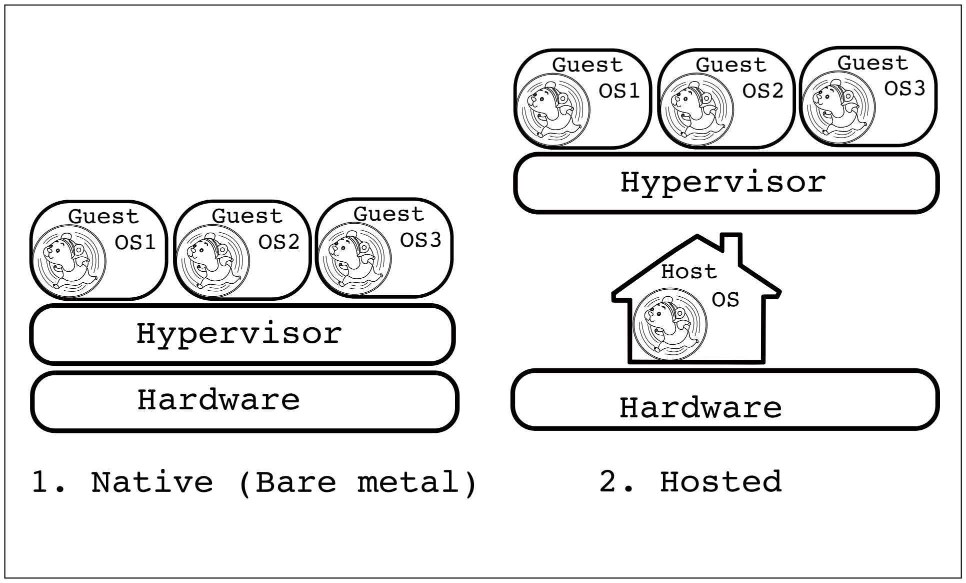
2.1.2 Virtualization

Virtualization has been a key enabling technology for the evolution of cloud computing into its current form. In particular, hardware virtualization has enabled IaaS providers to efficiently use the available hardware resources in order to provide computing and storage services to their clients.

Popek and Golberg defined a set of virtualization requirements in "Formal requirements for virtualizable third generation architectures" [18], which served as guidelines for the design of virtualized computer architectures. The authors have defined three properties of interest for a *virtual machine monitor* (VMM) also known as a *hypervisor*: **equivalence**, **resource control** and **efficiency**. This definition of hypervisors required satisfying all of the properties. In a later definition, Smith and Nair [19] only assume equivalence and resource control properties for VMMs, while *efficient VMMs* are required to satisfy all of the properties.

Figure 2.1 presents a classification of hypervisors according to Popek and Goldberg [18]. *Native* (or *bare metal*) hypervisors run directly on the host hardware, while *hosted* hypervisors run in the environment of an operating system (OS) and hence their access to the hardware resources is mediated by the OS.

Figure 2.1: Types of hypervisors according to Popek and Goldberg



Examples of native hypervisors are Citrix XenServer, VMWare ESX/ESXi and Microsoft Hyper-V hypervisors. KVM and VirtualBox are examples of hosted hypervisors.

2.2 Under the hood

While Amazon Web Services pioneered enterprise cloud computing [20] with its Amazon EC2 and Amazon S3, it has not established any well defined standard of cloud architecture and data exchange interfaces. As a result of several competing cloud computing projects that have either been released as open source projects or have been created as community-developed open source projects, currently there is range of cloud computing management platforms that are open for examination and implementation.

Thus, the currently available Open source cloud management systems are:

- *OpenNebula* has started as a European research project in 2005 and supports Xen, KVM and VMWare hypervisors. One of its main advantages is its flexible architecture that allows for multiple combinations of hardware and software platforms [21];
- *OpenStack* is an open sourced project based on the collaboration between National Aeronautics

and Space Administration (NASA) and Rackspace [3]. The project maintains compatibility with the Amazon EC2 interfaces and focuses on massively scalable, flexible cloud deployments.

- *Nimbus* is a scientific project that focuses on implementing and supporting features of interest for the research community. The projects offers a set of tools that allows its users to combine other platforms, e.g. OpenStack and Amazon EC2 [4];
- *Eucalyptus* is a community-driven open source project that aims to support wide compatibility with the EC2 interfaces in order to allow hybrid implementations that include both EC2 and Eucalyptus clouds [5].
- *other projects* that have a narrow specialization and smaller distribution include *Enomaly* [6], *Redhat Cloud*, Yahoo's *TrafficServer* and other smaller actors.

Along with open source IaaS implementations, there are a number of commercial products which are however out of the scope of this section. Rimal et al provide a thorough examination of the taxonomy of cloud computing systems as of 2009 [22], where they describe the main providers of cloud computing services and cloud computing platforms available at the time. Furthermore, Jim et al provide examples of well-known SaaS products (e.g. *Dropbox, Twitter, HeroKu*) that are deployed based on infrastructures maintained by commercial IaaS and PaaS providers [23].

2.2.1 Architectural overview of the OpenStack cloud management platform

OpenStack has been chosen as the implementation platform used to validate the solution explored in this thesis. The motivation behind the choice of OpenStack as the implementation platform is mainly based on the wide industry interest and active community participation. The motivation factors are covered in more detail below:

- *Industry interest and adoption:* currently OpenStack is supported by "more than 175 companies" [7]. Considering the scope and the aim of the thesis, support from Intel and AMD (which are also members of the Trusted Computing Initiative) was an important industry adoption factor.
- *Community interest:* since its first release in 2010, OpenStack has had a rapid community-driven evolution and is currently at its fifth release.
- *Availability of source code* – OpenStack source code is licensed under an Apache License, a permissive license which does not require the distribution of modified versions under the same license.

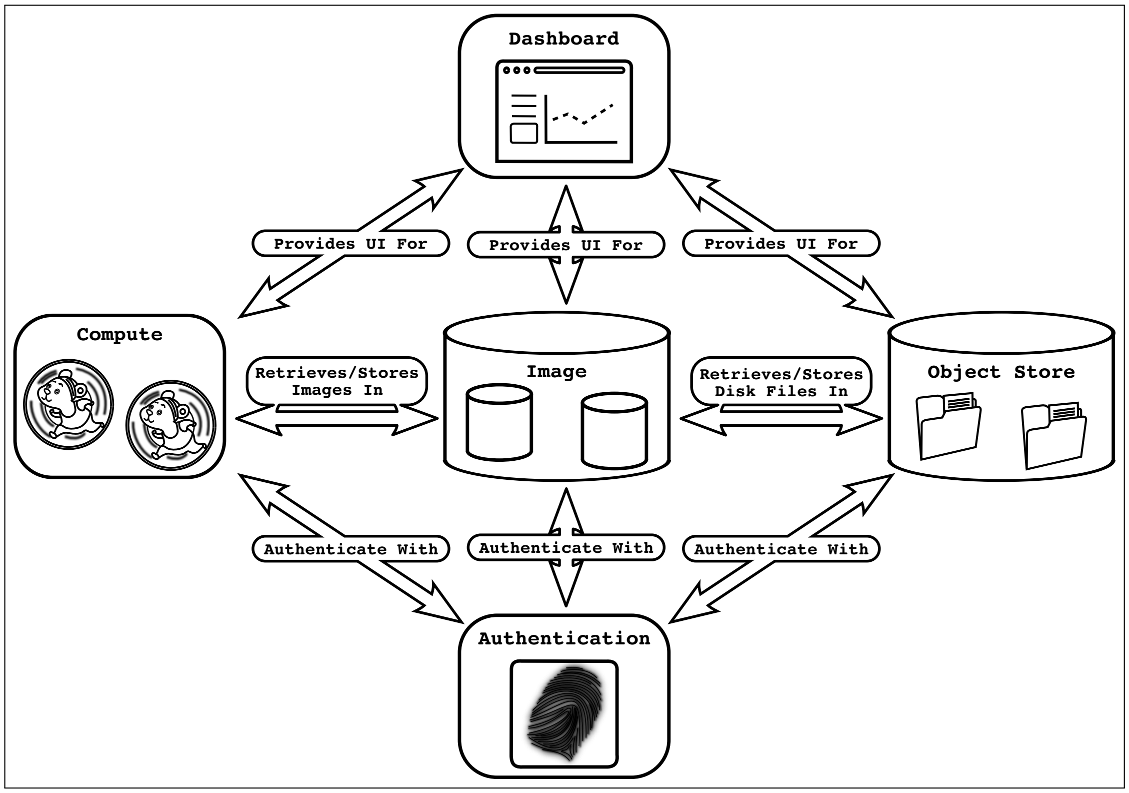
A brief introduction to the OpenStack platform is necessary in order to clarify the implementation of the secure VM launch protocol.

On a higher level, OpenStack is a collection of independent components that communicate with each other through public APIs and collectively form a robust cloud computing platform. From a logical view, also displayed in figure 2.2 [8], OpenStack is comprised of a *dashboard* which serves as a graphical user interface for the *compute component*, an *image store* and a *object store*. The three latter components authenticate through an *authentication* component.

The current release of OpenStack ("Essex") comprises five components which correspond to the above logical structure:

[3]The OpenStack project http://openstack.org/
[4] The Nimbus project http://www.nimbusproject.org/
[5]Eucalyptus Cloud http://www.eucalyptus.com/eucalyptus-cloud
[6]http://www.enomaly.com/
[7]http://openstack.org/community/companies/
[8]Concept courtesy of ken.pepple.info

Figure 2.2: Logical architecture of OpenStack



- *Horizon* is a Django-based dashboard which serves as a user and administrator interface to OpenStack. The dashboad is deployed through `mod_wsgi` in Apache and is separated into a reusable python component and a presentation layer. Keystone also uses an easily replaceable data store which keeps information from other OpenStack components.

- *Nova* is a core component of OpenStack and focuses on providing on-demand virtual servers. Nova offers several services, spawned on different nodes in an OpenStack deployment depending on the purpose of the node. The services are *nova-api, nova-compute, nova-volume, nova-network* and *nova-schedule*. Additional services, which are not part of nova but are however used by it are a queue serve (currently RabbitMQ is used, however any other queue system can be used instead) as well as a SQL database connection service (MySQL and PostgreSQL are supported for production, sqlite3 for testing purposes).

- *Glance* is VM image repository that stores and versions the images that are made available to the users initially or modified through subsequent runtime updates.

- *Swift* is an object store with a distributed architecture which aims to avoid single points of failure and facilitate horizontal scalability. It is limited to the storage and retrieval of files and does not support mounting directories as in the case of a fileserver.

- *Keystone* is a unified point of integration for the OpenStack policy, token and catalog authentication. Keystone has a pluggable architecture to support multiple integrations, and currently LDAP, SQL and Key-Value Store backends are supported.

The OpenStack documentation [9] offers detailed information about each of the above named components and their interaction.

2.3 Security Concerns

A monetization of the risks involved for the main assets that need to be protected (data, algorithms, activity patterns or business reputation) would show that each of the aspects is likely to have a different value for each organization or person. Hence, cloud users would benefit from both a choice of different *levels of security* based on their requirements as well as different aspects of security (e.g. special attention to business reputation risks). Both cases bring along their own trade-offs and implementation peculiarities.

In the given scenario, a constant research effort in the area of cloud storage and cloud computing security will help achieve the balance between economic feasibility, ease of deployment and a suitable collection of security considerations for each cloud service (CS) client.

2.3.1 Risk aspects of public cloud services

Along with the multiple economic, technological and management benefits of cloud computing services for organizations, there are a number of implementation risks that must be taken into account. The *Guidelines on Security and Privacy in Public Cloud Computing* published by NIST offer an overview of the security, privacy and availability risks of cloud computing [24]. The NIST guidelines identify, among other points, the following risks related to the use of cloud computing by organizations:

- *Governance* Due to their wide availability and in many cases high degree of usability, CS (especially on the SaaS level) can easily bypass the security, privacy and software use policies adopted by the organization. While ensuring that systems are secure and risk is managed is possible (although not trivial) in the case of in-house system deployments, that is far more difficult in the case of cloud services. One immediate reason for that is the fact that such services are made available through the public network, while their backends are running in unknown locations out of the security parameter of the organization. This can lead to a potentially vulnerable mix of secure and insecure services used throughout the organization.

- *Compliance* to laws and regulations in the case of CS is more difficult compared to in-house systems for several reasons, such as inability to ensure proper disposal of data, limited ability to control and ensure the geographic location of data, and restricted possibilities for electronic discovery of data in case of litigation.

- *Trust* Through the use of cloud computing and CS the organization relinquishes control over significant parts of aspects of security and privacy. As a result of this, the organization makes a commitment and places trust into the control mechanisms and processes employed by the cloud provider. One risk is the potential for insider access to the information, provoking both intentional incidents leading to loss or corruption of data, or unintentional errors, leading to massive unavailability of the CS. Another risk is the potential lack of clarity over data ownership, especially in border cases such as transaction data generated through the use of CS. Third, the fact that many CS are *composite*, i.e. themselves operating through combining or nesting other CS implies that the unavailability of either a horizontal or vertical component dependency would in many cases be propagated to user level. In case damage is inflicted as a result of service unavailability, the responsible party may be hard to identified, as pointed out in [25]. Fourth, *visibility* of the state of the system and the state of the data produced by the CS is crucial in the process of managing security and privacy risks. However, such visibility can be easily lost as a result of migration from a

[9]OpenStack Documentation Page http://docs.openstack.org

service deployed in-house to a CS. The cloud provider is likely to be resistant to direct audits of the state of its infrastructure and as a result a third party would need to be assigned for independent regular audits [24]. Fifth, transactional data generated in the process of CS utilization, although not important to the customer, can prove to be useful for social engineering attacks against the customer. In other scenarios transactional or ancillary information can be a threat to the privacy of the organization's customers (if exposed as a service for public use) in case it is sold or leaked. As a result, lack of a clear and explicit ownership of such metadata can pose a serious risk for the organization.

- *Operational aspects* The architecture of the CS model can contain a range of risks on both the CS server and client side. First, although virtualization offers additional security benefits through software isolation, increasing the attack surface is a risk in itself. The hypervisor can be compromised as well as the sensitive data contained in the customer's virtual machines can be leaked during VM launch, migration or paging. Secondly, the security of the virtual network which ensures connectivity between instances deployed in the cloud or between the cloud instances and the Internet must be taken into account. While traffic monitoring is important for intrusion detection, traffic between hosts on a virtual network might not be visible to network-based intrusion detection systems [26]. Third, ensuring the integrity of *virtual machine images* (VMI) loaded by the cloud provider remains an open issue. While a certain ability to verify the properties of the virtual machine can be built into the VM image by the customer in case of a tailored VMI, **not even such simple mechanisms are available in the case of generic virtual machines offered by the cloud provider**. Even in the case of a bona fide IaaS provider, malicious VMI can be contributed to the IaaS provider's image repository and maliciously imposed to the IaaS users [24] Fourth, on the client side secure key management presents an ever more complex process due to the proliferation of multipurpose handheld computing devices that are also used for cloud data access and management.

- *Data protection* From the CS customer perspective, there are fewer mechanisms for data protection when data is created through CS or maintained in cloud storage. Two aspects of data protection are considered, namely data availability and data access control. The first aspect depends on the migration and backup capabilities offered by the type of the CS chosen by the client. The second aspect is less trivial, due to the specifics of the *shared multi-tenant environment* in which CS are deployed.

  Thus, besides the fact that control and responsibility for the data is transferred from the data owner to the CS provider, physical isolation of data processing units is substituted by logical isolation in a multitennant environment. The type of CS (PaaS, SaaS, IaaS) used by the client determines both its degree of control over the underlying software stack and the type of logical data separation. For example, protecting commingled data [10] (in the case of SaaS) is more complex than collocated data [11] (in the case of IaaS) since on one hand the user has less control over the underlying software and on the other hand the complexity underlying a SaaS-level application increases the potential attack surface.

Beyond the ones mentioned above, other potential CS risks are related to identity and access management, software isolation, availability issues and incident response aspects. More details about these aspects can be found in [24].

[10] Data which is stored in a shared data store
[11] Data which is logically or physically separated

2.4 Trusted Computing

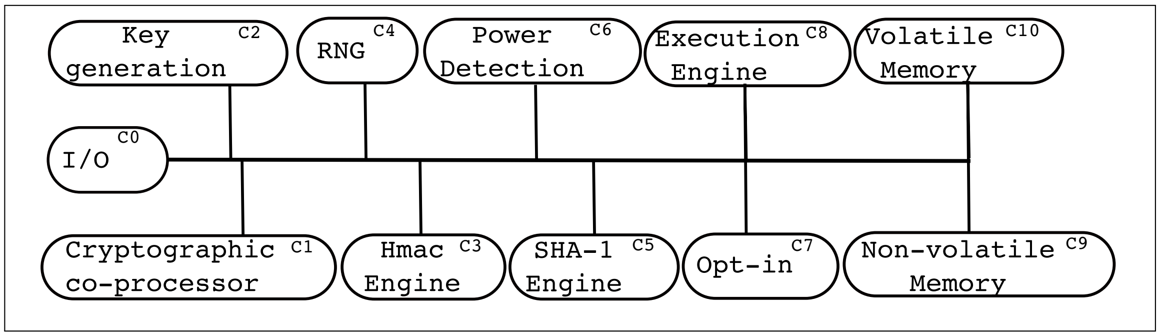
The trusted computing group (TCG) consortium [12] has developed specification for the trusted platform module (TPM), a hardware component that a cornerstone of Trusted Computing.

Trusted computing is a concept developed and promoted by the TCG and has at its core the ability to verify and enforce a certain consistent behavior using hardware means [27]. Furthermore, the TPM can be used to allow external parties to ensure that a certain host bearing the TPM is booted into a trusted state. That is performed by verifying the set of digests (called *measurements*) of the loaded software, successively produced throughout the boot process of the device. The measurements are stored in a protected storage built into the TPM chip and are therefore resistant to software attacks, although vulnerable to hardware tampering [13]. The structure and the working principles of the TPM are presented in the following sections.

2.4.1 TPM Structure

According to the TPM specification v.1.2 [29], the TPM chip contains 11 discrete standardized components, displayed in figure 2.3.

Figure 2.3: Trusted Platform Module Component Architecture



- C0: *Input/Output*, which performs protocol encoding and decoding, as well as directs the information flow over the communications bus;
- C1: *Non-volatile Storage* is a persistent storage that is used to store the non-migrateable keys – Endorsement Key (EK) and Storage Root Key (SRK) – as well as owner authorization and persistent configurations.
- C2: *Platform Configuration Registers (PCR)* can be implemented in either volatile or non-volatile storage. The TCG specification [29] prescribes at least 16 PCRs, where PCR 0-7 are reserved for internal TPM use and registers 8-16 are available for OS and userspace application use.
- C3: *Attestation Identity Keys (AIK):* This component stores the persistent keys that are used to sign and authenticate the validity of the information provided by the TPM in the case of external

[12] Trusted computing group http://www.trustedcomputinggroup.org/

[13] the IBM 4758 secure co-processor does have some protection against hardware tampering, however in order to reduce the costs of such secure co-processors the TPM specification was developed for a simplified version only resistant to software attacks [28]

attestation. AIK can also be stored in encrypted form in an external data store, in order to accommodate multiple users on the same platform.

- C4: *Program code* contains the firmware that is used in order to measure the platform devices and is the representation of the core root of trust measurement (CTRM).
- C5: A *Random number generator (RNG)* is implemented in the TPM in order to assist in key generation;
- C6: A *SHA-1 engine* is implemented for hash generation to assist in signature creation;
- C7: *RSA key generation* is a component to create asymmetric encryption keys based on the Rivest, Shamir, Adelman protocol [30].
- C8: *RSA engine* is used in order to perform the signing, public-key encryption and decryption operations based on the RSA algorithm.
- C9: The *Opt-in* component allows to maintain the activation state of the TPM chip, the possible states being *enabled, disabled, deactivated.*
- C10: The *Execution Engine* is a component that executes the operations prescribed by the logic in the *program code.*

2.4.2 TPM operations

Using the above described components, the Trusted Platform Module provides a set of standard unmodifiable functionality pre-defined by the hardware vendor according to the TPM specification. Several functions that are important in the scope of this thesis are:

- The *Integrity measurement* function enables the TPM to collect the measurement digest of the operational state of the system. Such measurement digests are stored in the PCRs and all subsequent digests are stored by *extending* the hash according to the formula

  ```
  PCR[n+1] = SHA-1(PCR[n] || Measured Data)
  ```

  The PCR values are discarded and rewritten on each system reboot.
- The TPM provides *Integrity reporting* functionality in order to report and *attest* the authenticity of the stored PCR values for external attestation purposes. This is done by digitally signing the *integrity reports* based on the PCR values with an AIK generated and stored in the TPM. The process of AIK generation is covered in detail in [29].
- As an endpoint of communication, the TPM provides a set of operations for secure message exchange based on the use of asymmetric cryptography, with the keys being generated and managed by the TPM. The four operations provided by the TPM are described in Table 2.1.

Table 2.1: Operations for secure message exchange provided by TPM

Binding	TPM offers protection of the message by means of asymmetric cryptography using encryption keys generated and maintained by the TPM. Thus, a message encrypted using a particular TPM's public key is decryptable only by using the private key of the same TPM.
Signing	functionality is implemented according to the same principles of asymmetric encryption described in [30].
Sealing	a special case of the binding functionality, where the encrypted messages produced through binding are only decryptable in a certain platform state (defined through the PCR values) to which the message is *sealed*. This ensures that an encrypted message can only be decrypted by a platform which is in a certain prescribed state.
Sealed-sign	offers the possibility to verify that the platform producing the signature is in a certain specific configuration. The verification is based on the measurements from a set of pre-determined PCRs that are evaluated to determine whether the platform has the required configuration.

Implementations

Having covered several essential points of the TPM specification, we visit the state of the art with regard to the support for trusted computing technology from the main server and PC manufacturers, namely Intel and Advanced Micro Devices (colloquially known as AMD).

Intel has implemented support for the TPM through the *Trusted Execution Technology (TXT)*, which includes a set of hardware enhancements that allow the creation of multiple separate execution environments [31]. A recent contribution towards trusted computing from Intel is the OpenAttestation Software Development Kit.

AMD has developed "AMD Presidio" [32], a technology similar to Intel's TXT. However, AMD Presidio only offers rudimentary support for TPM.

2.4.3 Applications and extensions

Several applications are available to provide integration of the TPM hardware with other systems as well as provide a wider functionality based on the capabilities of the TPM.

Figure 2.4 contains a visual representation of the current trusted computing software stack.

Starting with the lowest level, the following hierarchy of TPM applications is available:

1. *TrouSerS* – a trusted computing software stack developed by IBM and released under the common public license, supported on i386 GNU/Linux. Several *nix distributions are supported such as SuSe, Fedora Core and RHEL, however functionality for specific releases requires ongoing testing [14].

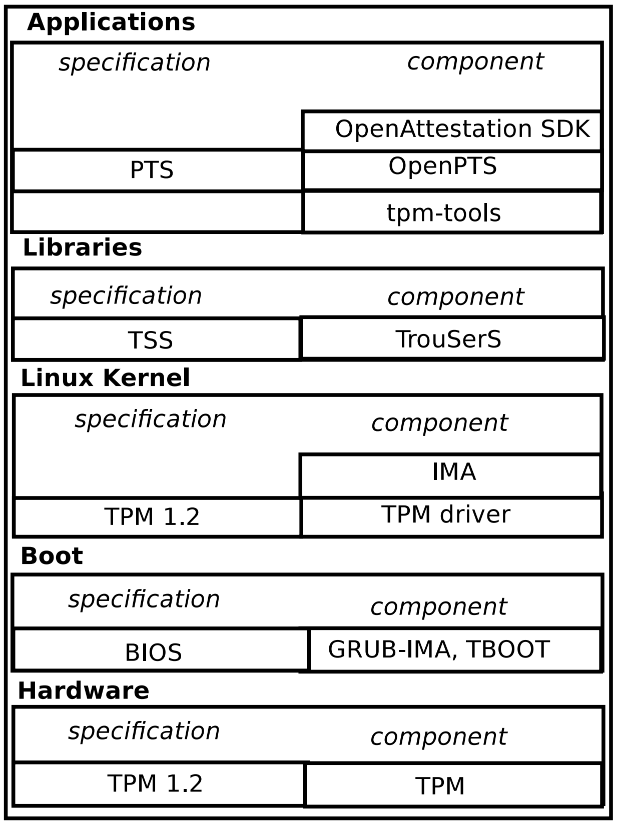
2. *Linux-IMA* – first introduced in the linux kernel version 2.6.30, integrity management architecture (IMA) maintains a runtime measurement list and an aggregate ingrity value over the list (if anchored to a TPM). Linux-IMA may or may not be enabled by default on specific distributions [15].

3. *Grub-IMA* – an enhancement of the Linux bootloader supporting TCG-compliant platforms enabled with TPM versions 1.1 and 1.2. The Grub-IMA patch supports GRUB version 0.97 and is

[14]TrouSerS project documentation http://trousers.sourceforge.net/faq.html

[15]Linux-IMA documentation http://sourceforge.net/apps/mediawiki/linux-ima/index.php?title=Main_Page

Figure 2.4: TPM applications stack



available for RHEL/CentOS, Fedora Core and Ubuntu [16]

4. *OpenPTS* – Open Platform Trust Services is a software stack that allows the verification of integrity of TPM-enabled platforms as well as their conformance to certain specific trust models. OpenPTS relies on TrouSerS, Linux-IMA and Grub-IMA for data collection [17].

5. *Intel OpenAttestation SDK* – in April 2012 Intel has launched [18] the OpenAttestation project [33], a software development kit that enables creation of cloud management tools with the capability to establish host integrity information. This functionality is supported by the TPM attestation procedure, based on the measurement values stored in the TPM's PCRs. The project supports integration with Intel TXT.

2.4.4 TPM Use Cases

The trusted computing group defines four initial use cases for the trusted platform module in the TCG specification architecture overview [29]:

- TPM can be used to reduce the risk to information assets and to facilitate risk management by offering additional protection to the credentials used to verify and assert the integrity and

[16]Grub-IMA project documentation http://ko.sourceforge.jp/projects/openpts/wiki/GRUB-IMA
[17]OpenPTS documentation http://ko.sourceforge.jp/projects/openpts/wiki/OpenPlatformTrustServices-0.1
[18]OpenAttestation open Source Launch: http://permalink.gmane.org/gmane.comp.boot-loaders.tboot.devel/338

authenticity of data. This is achieved by using the TPM to store keys, hash values and other secrets in a protected hardware storage. Restricted access data is thus protected from software attacks and can only be obtained through hardware tampering.

- As a facilitator of *asset management*, TPM can be used as part of the functionality to assert the ownership of the platform while also allowing *certain* third parties to have access to the data protected by the TPM.
- TPM chips could also be used in the context of e-commerce transactions in order to maintain the context of a secure and verified transaction throughout future transaction without the need for repeated verification of the client's and vendor's integrity.
- In the context of security monitoring and emergency response, the TPM provides the capability to correctly report or verify the state of the system. This can be used to e.g. identify the systems that require an update or to identify compromised systems.

Examples of a commercial product build on top of the Trusted Platform Module is Microsoft's BitLocker [19] or HP ProtectTools [20]. BitLocker is a full disk encryption functionality available with certain versions of the Microsoft Windows operating system [21], which allows to protect data from unauthorized inspection. HP ProtectTools is a tool for security management implemented using HP proprietary technology, which uses the trusted platform module for key management [22].

While the use cases defined by the TCG are broadly defined to encompass multiple applications of the TPM, the current thesis creates a a new special use case for the TPM and Trusted Computing by placing them in the context of cloud computing. Hence, we consider a new use case for TPM components in commodity hardware, namely:

> *Provisioning of trusted VM instances in IaaS public clouds in order to ensure a trustable computing environment for CS customers.*

This use case is the focus of the current thesis and will be extended in the following chapters.

2.5 State of the Art in Public Cloud Computing Security

2.5.1 Cloud management interface

One of the fulfilled "promises" of cloud computing is the reduction of the upfront hardware purchase and setup costs. Thus, the client running virtual machine (VM) does not need to bear the overhead of managing of the location and the hardware aspects of the server that is being used. Instead, a web administration console is the single interaction point between the cloud service provider and the image administrator [34]. The web administration console offers a wide range of administrative controls, such as creating instances, generating SSH keys for the instances, starting and stopping VMs, as well as commissioning other cloud services. In this situation, the authentication framework employed by the cloud provider is the only safeguard between malicious actors and the control of a cloud user's account. The web accessibility of the administrative console makes it vulnerable both due to the sheer accessibility and to its potential susceptibility to web-based attacks such as cross-site scripting [35], phishing attacks [4], DNS cache poisoning, session hijacking etc.

Furthermore, as demonstrated by Somorovsky et al in a recent security evaluation of the Amazon Web Service (AWS) [1] a potential attacker can obtain full control of a cloud user's management console and hence of the whole cloud project belonging to the respective client.

[19] MS BitLocker: http://technet.microsoft.com/en-us/library/cc766015(v=ws.10).aspx

[20] HP ProtectTools, http://h20331.www2.hp.com/Hpsub/cache/292230-0-0-225-121.html

[21] "What's new in BitLocker on Windows 8", http://technet.microsoft.com/en-us/library/hh831412.aspx

[22] HP protect tools http://h20331.www2.hp.com/Hpsub/cache/292230-0-0-225-121.html

Attack

In their analysis, Smorovsky et al relied on both novel signature wrapping attacks and cross-site scripting vulnerabilities in order to gain control of the Amazon EC2 and S3 services. Cross-site scripting attacks are based on exploiting the relationship between a web client and a server considered trusted, which makes it possible to execute an injected script on the client side with the server's privileges. In [36], the authors define three factors facilitating the wide distribution of XSS vulnerabilities, namely predominance of web applications that display untrusted input, availability of an unsafe default for passing untrusted input to the client in most web application programming languages, and the availability of multiple, often browser specific ways of invoking the JavaScript interpreter.

In the case of signature wrapping attacks, the authors used the meaningful and informative SOAP message responses to check the possibility of the signature wrapping attack. Using a single eavesdropped *MonitorInstances* SOAP request the authors were able to start new VM instances, stop any running instances or create new images and gateways in a victim's cloud environment [1]. The authors remark that the use of SSL/TLS is not a solution to signature wrapping attacks, since signed SOAP messages can be traced by other means.

In the case of the identified cross-site scripting (XSS) vulnerability, the authors used a download link vulnerability in the Amazon store website in order to gain control over the cloud management interface.

Countermeasures

Below is a selection of the countermeasures research lines with respect to signature wrapping attacks examined by the authors of the attack. The selection includes:

- Validating each message against an appropriate security policy; however, the countermeasures were considered ineffective;
- XML schema validation, which can detect SOAP messages modified by a signature wrapping attack. However, this operation carries a significant performance hit and is not performed by the standard web frameworks.
- Providing and signing additional information on the structure of the SOAP request has also been suggested; however, this approach can also be circumvented using the extensible and flexible structure of SOAP.
- The countermeasure suggested by the authors themselves is a stronger isolation between the signature verification functionality and the business logic of the application.

2.5.2 Challenges in securing the virtualized environment

A significant body of research has been carried out in the area of secure virtual machines and particularly confidentiality of virtual machines in untrusted environments. Below is an overview of publications which explore the possibility to secure the integrity of the VM and its environment.

Focus on VM isolation

Kong attempts to secure the VM in a XEN virtualization environment where Dom0 is susceptible to both internal and external attacks (i.e. is generally unadjustable) in [4]. Five components of the architecture are considered:

- During the system boot phase the author examines the challenge of building a trusted computing base from the platform power to BIOS to the boot loader and finally the hypervisor. In order to tackle that, the author proposes enhancing the BIOS with a *"Core Root of Trust for Measurement"* (CRTM) that will be first component to be loaded.

- Direct Memory Access is used with IO Memory Management Units in order to isolate the devices and only allow access to certain memory ranges assigned to each VM;
- A virtual TPM is instantiated and placed in a specialized domain Dom0 U and further executes the role of the TPM for other Dom0 X.
- The guest VM boot process uses a custom protocol which relies on asymmetric cryptography in order to ensure the VM is instantiated in a trusted environment;
- In the process of saving and restoring the image the authors adopt a "per page encryption method" [4] to encrypt the instance before it's handed over to Dom0.

A major drawback of the proposed method is a reported tangible performance hit. However no clear benchmarks have been provided. Furthermore, secure VM migration is left as an unsolved challenge.

Minimizing the TCB

As minimizing and sealing the attack surface is one of the basic principles of application security [37], several of the examined approaches aim to solve this by reducing the trusted computing base.

Li et AL address the issue based on the assumption that in a virtualized environment the management OS should not be trusted [38], The authors propose the architecture with a reduced TAB which excludes the management OS and ensures a secure runtime environment for the guest VM. In the proposed solution Dom0 is able to manage the domains without knowing their contents, with the architecture providing a secure run-time environment, network interface, and secondary storage for a guest VM

The authors consider the scenarios when a Dom0 can subvert the VM to insert root kits or other external code, to undermine the integrity of the new VM execution environment by setting up wrong initial page tables, corrupt the new VM execution environment, and maintain foreign mappings of the new VM memory area in order to read the memory area during the VM runtime.

The solution considers several steps for executing the proposed solution:

- During the domain building and shutdown a malicious Dom0 could potentially subvert the launched VM in a number of ways [38]. In order to mitigate this the authors propose to modify the hypervisor to perform foreign mapping cleaning, to check that none of the pages that allocated to the new VM are pointing to Dom. In addition to that an integrity check for the new VM kernel and the CPU context is performed during the Dom0 hyper call to the supervisor to check the integrity of the VM before launching.
- With regard to the domain run-time the authors analyze the types of hypercalls used within the communication between Dom0 and Dom0 U and specifically consider the hypercalls that are harmful to the privacy and security of Dom0 U but necessary for its management. The suggested solution includes pushing hypercalls related to memory and CPU calls into the hypervisor.

Similar to some previous solutions, the proposed architecture involves a significant overhead, which is however considered by the authors acceptable since the tasks incurring the overhead are not performed continuously.

A more radical step towards the minimization of the TAB is taken by Szefer et AL who discuss eliminating the hypervisor attack surface [39]. Authors propose a virtualization architecture that does not assume the presence of a hypervisor, except for the VM creation and boot stages. Instead of relying on the hypervisor for arbitration of resources between VMs, the task is performed by mechanisms built into the hardware (e.g. hardware paging mechanisms for memory isolation, or virtualization capabilities in some hardware devices (like NICs)). The key ideas of the proposed NoHype architecture are *pre-allocating memory and cores, using only virtualized I/O devices, short-circuiting the system discovery process and avoiding indirection.* A prototype implementing the proposed architecture based on Xen

4.0, Linux 2.6.35.4 and commodity Intel hardware offers a 1% performance improvement compared to the use of a Xen hypervisor, except for `gcc` benchmarks. It is important to note however, that the use of a hypervisor is not fully excluded in the architecture proposed by Szefer. While security risks can potentially decreased by reducing the role of the hypervisor to VM creation and boot, the authors do not mention how a compromised hypervisor can be prevented from performing malicious actions and compromise the integrity of the VM during the stages when it is employed.

2.5.3 The trust in TPM

Multiple research projects aiming to secure the VMs in a cloud environment heavily rely on the TCG TPM can be found in [6, 40, 5, 41], as well as the more recent [10, 14]. All of the above papers use the TPM component in order to provide a reliable root of trust that could be used to either bootstrap the rest of the software stack on a TPM-enabled machine or obtain certain guarantees regarding the integrity or the policy compliance of the target host when migrating a VM instance to a different host. Some of the above publications will be examined closer to identify their contributions towards addressing the problems expressed in PROPOSITION 1 and PROPOSITION 2, formulated in chapter 3.

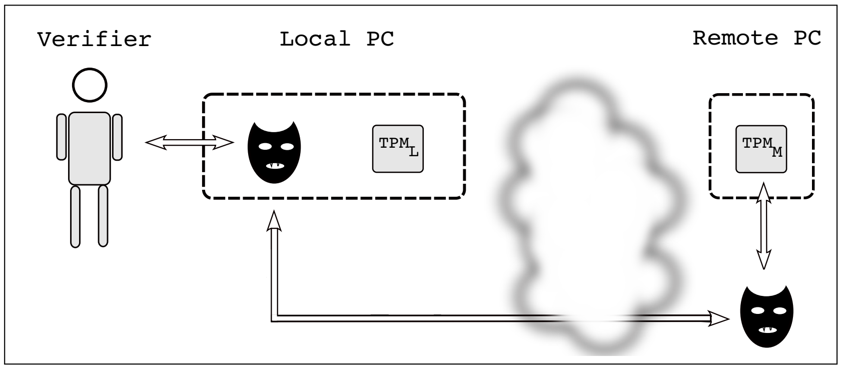
A *reliable* and *trustable* TPM indeed offers a wide range of capabilities to enhance the security of virtualization environments, in particular and cloud computing environments in general. Parno et AL has examined the current solutions with regard to provisioning of a *root of trust.* The authors distinguish four types of solutions, namely general-purpose devices with significant resistance to physical tampering, general-purpose devices without significant physical defenses, special-purpose minimal devices, and research solutions that attempt to instantiate a root of trust without custom hardware support. IBM 4758 is an example of a full-blown general purpose tamper-resistant and tamper-responding cryptographic co-processor, which includes capabilities to respond to hardware tampering in several ways, e.g. erasing the data or disabling the device [28]. TPM on the other hand, is a general-purpose device without significant physical defenses, and has been designed to be secure only against *software* attacks. TPM provides sealed protected storage, rather than tamper-responding protected storage. Thus, a TPM physically accessible by an adversary without physical audit capabilities on the client side can not be trusted [42, 11]. Alas, one of the core driving forces behind today's rapid development of cloud computing is the reduction of the hardware maintenance overhead by purchasing computation storage capabilities from cloud providers who maintain the hardware infrastructure and hence have full access to the servers, including the TPM. That opens the door to a number of additional potential vulnerabilities. The above-named hardware attacks have been discussed earlier [43, 44] and are out of the scope of this paper. Instead, we will focus on some attacks against the TPM in general and the Intel Trusted Execution Technology (which relies on the TPM).

In [11], Parno introduces the concept of *cuckoo attack* (Fig. 2.5), where malware on the user' machine forward the calls intended for the local TPM to a remote attacker who feeds the messages to a TPM controlled by an adversary.

The physical control of the machine allows to violate the security of the TPM via hardware attacks and thus control all communication between the verifier and the local TPM. In this scenario, the attacker will be able to access all of the answers the local TPM is expected by the TPMs manufacturer confirms that the TPM is genuine and is used to endorse the public key of the TPM. However, it only endorses the fact that the public key belongs to *some genuine* TPM and not the *specific* TPM [11]. A potential solution would be to corroborate the public key provided by the TPM with the public key provided by the manufacturer, however this raises issues regarding secure distribution of TPM public keys by the manufacturers.

A more vendor-specific attack is discussed in a whitepaper by Wojtczuk and Rutkowska, where the authors discuss the vulnerabilities in the design of Intel's *Trusted Execution Technology*, which is Intel's response to the trusted computing trend [45]. The attack that allows to subvert the Xen hypervisor [46] is based on two vulnerabilities identified in the System Management Mode, which on x86x86_64

Figure 2.5: Cuckoo attack implementation model



architectures is more privileged than ring 0 mode and a hardware hypervisor (VT/AMD-v). While the specific bugs described in the paper are likely to be patched by the chipset manufacturer, the paper raises a number of questions regarding the security considerations of the TXT architecture.

2.5.4 Challenges and further research

A review of research challenges within cloud computing reveals issues on the whole spectrum from the hardware TPM security to the higher level-aspects of IaaS and PaaS cloud computing security. Recent papers on research trends within cloud computing security include Chen and Katz' assessment of security challenges that are *specific* to cloud computing security [4], Hay et al's paper on security challenges on IaaS cloud computing[47] and Zhang et al's broader assessment of the state of the art and research challenges within cloud computing [48].

Auditability

The currently unsolved and increasingly important issue of mutual auditability is emphasized in both [48] and [4]. Zhang et AL suggest that the remote attestation capabilities enabled by the TPM could be used for attestation on single machines, however they are not enough due to the dynamic migration of VMs between machines in the cloud environment. This is addressed by the solution proposed by Santos et AL, which includes a fixed set of attested trusted machines within the cloud could be extended to enable auditability even in the event of VM migration within the cloud.

Periodic probing of certain functions, files or licenses presents special case of auditability and is specifically relevant for specialized clouds, e.g. telecommunication services clouds [49]. Aspects of this task could be addressed by applying some principles of the VM introspection based architecture for intrusion detection [50] described by Garfinkel and Blum. In this case the VMM-based intrusion detection component described in the original paper could be reconfigured to periodically take a set of VM probes that have been agreed upon by both the cloud host and client. This could complement TCG-oriented solutions aimed at ensuring trusted virtualized environments.

Reducing the trusted computing base

As mentioned above, several authors ([38, 39]) have researched the possibility of improving the security of virtualized environments by reducing the trusted computing base. Reduction of the trusted computing base is likely to reduce the potential for software vulnerabilities as more of the functions currently performed by the hypervisor could potentially be implemented increasingly closer to the mechanisms provided by the underlying hardware. Given the results by Li [38] and Szefer [39], this is a promising research direction. While reduced functionality and some performance hits might arise as potential challenges on the way, some solutions could be used for specialized clouds with less strict performance or functionality requirements yet higher security demands.

New approaches in avoiding the cross-VM attacks

As Restenpart et AL mention in [51], preventing cross-CM attacks in the case of VMs placed on the same physical machine is difficult and in their view can can not be executed other than by placing the VMs on different physical servers. Despite the drastically increasing costs in case of such a solution, it might be applicable to certain types of specialized clouds. The question in this case is how could placement on different physical machines be leveraged to implement additional mechanisms to enhance the security of such a virtualized environment.

Minimizing the impact compromised TPM private keys

The question of certificate revocation is mentioned by Garfinkel et AL in [52]. However, while discussing hardware revocation the authors consider only the case when the private key of a specific TPM chipset has been compromised. The authors do mention that the compromise of the manufacturer's signing key would have much more serious consequences without providing any further solutions. Given the wide acceptance of TPM modules and the large number of research projects heavily relying on TPM on one hand and the recent cases of root CA breaches [53] on the other hand, an exposure of a manufacturer's signing keys can not be excluded. Given the difficulties of revoking certificates embedded in the TPM chips, it is necessary to explore the available options that could be used in the event of manufacturer signing keys exposure.

2.6 Related work in launch and migration of virtual machines

2.6.1 Trusted virtual machines and Virtual machine managers

Santos et AL propose a design of a trusted cloud computing platform (TCCP) that ensures VMs are running on a secure hardware and software stack with a remote and untrusted host [5]. The platform is comprised of three trusted components, namely a *trusted virtual machine monitor*, *trusted coordinator (TC)* and an *external trusted entity (ETE)* and an untrusted *cloud manager*. The TCCP allows a client to reliably and remotely attest the platform at the cloud provider's location to verify that it is running trusted VMM implementation and thus make sure that the computation is running in a guest VM is secure.

The usual attestation process involves compiling a *measurement list* (ML) that contains an array of hashes of the software stack involved in the boot sequence. The ML is securely stored in the machine's TPM and can be subsequently used for remote attestation using a protocol based on asymmetric encryption using the TPMs private-public keypair. However, this attestation mechanism does not guarantee that the measurement list obtained by the remote party corresponds to the actual configuration of the host where the VM has been running (or will be running in the future).

To circumvent that, the authors propose an enhanced attestation process where a *trusted coordinator* (TC) stores the list of attested hosts that run a TVMM and which can securely run the client's VMs. Each of the trusted hosts maintain in their memory a *trusted key* (TK) that is used for identification, each time the client instantiates a VM on the trusted host. Finally, the TC is hosted by an *external trusted entity* which is out of the reach of the cloud provider and hence can not be tampered with (here the authors make a parallel to the root CAs we find in the Public Key Infrastructure (PKI) model).

The paper presents a good initial set of ideas for secure VM launch and migration, in particular the use of an external trusted entity. However, a significant drawback of this solution is the fact that the TK resides in the memory of the trusted hosts, which leaves the solution vulnerable to cold boot attacks [54] where the key can be extracted from memory. Furthermore, the authors require that the ETE maintains information about all of the hosts deployed on the IaaS platform, but does not mention any mechanisms for anonymizing this information, making it valuable to potential attackers and unacceptable for a public cloud service provider. Finally, the paper does not mention any mechanism for revocation of the TK, nor any considerations for the re-generation of TKs outside of host reboot.

2.6.2 "Seeding Clouds With Trust Anchors"

A decentralized approach to integrity attestation is adopted by Schiffman et AL in [55]. The primary concerns addressed by this approach are the limited transparency of cloud computing platforms and the limits to scalability imposed by third party integrity attestation mechanisms, as described in [5]. Thus, the role of a third party host attestation component is fulfilled by a proposed 'cloud verifier' (CV). The authors examine a trusted cloud architecture where the integrity of the cloud hosts (termed 'network controller in the paper) is verified by the IaaS client through the CV, which resides in the application domain of the IaaS platform provider. Integrity verification is performed based on the measurements produced by the TPM component of the CV host. Thus, in the first step of the protocol *the client verifies the integrity* of the CV in order to include the CV intro its trust perimeter, if the integrity level of the CV is considered satisfactory. In the second step of the protocol, the CV sends attestation requests from the node controllers (NC) where the client's VM will be launched, thus extending the trust chain to the NC. Next, the NC verifies the integrity of the VM image, which is countersigned by the CV and returned to the client which evaluates the VM image integrity data and allows or disallows the VM launch on the NC.

While the idea of increasing the transparency of the cloud computing platform for the client is indeed supported by the industry ([3, 4]), the authors do not clarify how the introduction of an additional integrity attestation component in the architecture of the IaaS platform has any positive effect on the transparency of the cloud platform. Furthermore, the proposed protocol increases the complexity model for the IaaS client with two aspects. First is the necessity to evaluate the integrity attestation reports of the CV and NC. Second is introduction of additional steps in the trusted VM launch, where the client has to take actions based on the data returned from the CV. This requires either human interaction or a fairly complex integrity attestation evaluation component (or a combination thereof) on the client side, something which will not contribute to a wide-scale adoption of the solution.

2.6.3 "Securely Launching Virtual Machines on Trustworthy Platforms in a Public Cloud"

Aslam et AL [56] examined the case of trusted VM launch in public IaaS platforms. The paper explores the scenario when a client employs virtual machine images provisioned through a *procurement server* by an IaaS provider and focuses exclusively on the launch stage of the CM lyfecycle in an IaaS platform.

The proposed model assumes a trusted computing base (TAB) up to the level of Management VM (known as Dom0 in the Xen VMM model) and relies on the use of TPM hardware for key generation and platform integrity (the protocol is also visualized in appendix D). In order to ensure that the VM instance requested by the client is launched on a host with verifiable integrity, the client encrypts the VM image (along with all the injected data) with a symmetric key which is *sealed* to a particular configuration of the host (reflected through the values in the PCRs in the TPM deployed on the host). To do that, the client must connect to the cloud service provider's network through an IPSec tunnel and negotiate a host that satisfies the client's security requirements. As a result, the client receives the means to directly communicate with the destination host (where the VM instance will be deployed), as well as TPM-signed data certifying the integrity of the host. Further, prior to that the client must attest the destination host platforms which are presented as trusted by the cloud service provider, and generate an asymmetric bind key that can be used to decrypt the VM image provided by the client. As a result, the VM image is only accessible by the host when in a certain *trustable* state. The protocol satisfies requirements regarding authentication, confidentiality, integrity, non-repudiation and replay protection.

The solution proposed by Aslam et AL presents a model which is suitable in the case of trusted VM launch scenarios for enterprise clients. It requires that the VM image is pre-packaged and encrypted by the client prior to the IaaS launch. However, a plausible scenario like the launch of a generic VM image made available by the IaaS provider is not covered in the proposed model. Furthermore, we believe that in order to decrease the number of barriers in the adoption of a trusted cloud computing model, as few actions as possible should be required from the client side. Likewise, direct communication between the client and the destination host as well as significant changes to the existing VM launch protocol implementations in cloud computing platforms hamper the implementation of such a protocol and should be avoided. The current thesis reuses some of the ideas proposed in [56], adopting however a more lightweight approach in terms of required user actions and requirements towards the launched VM image and the cloud computing platform.

CHAPTER 3

Problem Statement and Scope

As pointed out in chapter 2 based on the results found in [4, 5, 6], concerns about the lack of clarity regarding data protection, reputation fatesharing, lack of traceability and transparency within cloud services as a whole, as well as the algorithms behind handling of VM images in particular are among the barriers that hamper the adoption of cloud computing in industry. However, while these barriers are indeed numerous, there is no reason to believe they are unsurmountable.

In this chapter we examine several problematic issues related to cloud computing which if solved, could potentially have a positive impact on the adoption of cloud computing in general and IaaS in particular.

3.1 IaaS security aspects revisited

Of the three main types of cloud computing described earlier (IaaS, PaaS, SaaS), IaaS offers the broadest customer control over the computing stack. Such broad customer control (and hence transparency, from the customer's point of view) provides the tools to address several of the concerns regarding adoption of public cloud computing services, namely "traceability and transparency within cloud computing", as well as "lack of information about the algorithms behind handling of VM images".

3.1.1 Control over the cloud computing platform

As defined by NIST, the context provided by IaaS offers customer access over the network to a sandboxed environment of a VM instance, or a collection of VM instances with limited control over the inter-VM network communication and no control over the underlying components of the cloud computing environment, such as the VM manager (or *hypervisor*), physical servers (or *hosts*) that support multi-tenant environments and the network communication between the physical hosts. Hence, the customer does not have any control over the whole software stack underlying the virtualized environment. While different hypervisor models treat the instructions from the VMs their own specific ways, a hypervisor (regardless of its type) is in a position to intercept and interpret the instructions passed from the VM instance to the CPU [57].

As a result, a compromised hypervisor can leak information about the data processed by the VM instance to the cloud platform provider or a malicious third party. Likewise, the host where the VM instance is running can be compromised by other software attacks, either by a malicious third party or the cloud service provider itself in the face of an insider (not necessarily malicious, as pointed out in [24]).

This results in the following **PROPOSITION 1:**

> In the current public cloud computing model, the IaaS user has no control over the choice of the integrity configuration of the platform where their VM image is launched. We state that it is possible to provide more granular control over the stack underlying the virtualized environment and enable the client to decide whether a certain operation should or should not be performed in a IaaS environment based on information about the structure and integrity of the underlying software and hardware stack.

3.1.2 Need for transparency and information

Another factor preventing the wider adoption of IaaS is that it is seen as a "black box" in terms of information about other VM instances collocated on the same physical servers, malicious attacks as well as intra-cloud migrations of the VM instance between different hosts.

Importance of awareness of other VM instances collocated on the same physical servers has been demonstrated by Ristenpart et al in [2], who describes an exploratory attack on Amazon Web Services. The authors have succeeded in creating a map of the placement of physical nodes in the Amazon cloud as well as map them to the live, running instances. Furthermore, by exploiting the Amazon placement algorithms and checking co-residence [1] based on Dom0 IP addressed, the authors have succeeded to migrate a malicious VM instance to the same host as the target VM instance. Co-residency with a target VM instance can be used for side-channel attacks as described in [38, 58, 39], making information regarding intra-cloud instance migration particularly important for bona fide customers.

Providing full information regarding placement and co-residence state of VM instances to IaaS customers would potentially enable them to take more accurate, dynamic decisions regarding trustworthiness of the IaaS VM instance. Therefore, in order to simplify the task we consider a subset of such information, namely assurance regarding trustworthiness of the underlying software and hardware (SW/HW) stack. This will be a less disruptive first step towards adding more details to the black box perspective of IaaS that is shared by the users of public cloud services.

Based on the above we formulate **PROPOSITION 2:**

> In the current public cloud computing model, it is not possible for a IaaS user to obtain guarantees regarding the integrity of the platform where the VM image is launched. Furthermore, there are currently no mechanisms for a IaaS user to verify the veridicality of the fact that a certain VM instance has been launched using the unmodified VM image provided by the user, unless the VM image has certain irreproducible and verifiable properties. We state the it is possible to provide the IaaS user with guaranteed, veridical and verifiable information about the integrity of the host running the client's VM instance, as well as guarantees about the veridicality of the VM instance.

3.2 Problem statement

Chapter 2 and the above sections in the current chapter have discussed some of the security aspects of public clouds and the risks related to adoption of public cloud computing, as well as the public opinion stance towards adoption of public cloud computing.

Two propositions have been formulated, regarding control over placement of the VM instance with respect to the integrity guarantees of the host running the client's VM instance.

The problem formulated above will be addressed by this thesis in the context of a specific use case. That will help reduce the complexity of the addressed question and allow us to focus on the exact issue with a minimum number of complementary aspects.

[1]Co-residence in this context is the situation when two VM instances are located on the same host

3.2.1 Specific use case considered

In this paper we consider the aspects of secure launch of generic VMs (VMs) in an untrusted **public cloud computing environment**. In this context, by *generic VMs* we mean the VMs made available by the cloud service provider but assumed to be identical with the vendor-issued models[2].

The scenario implies that the actor that launches the VM instance (further referred to as *"client"*) requires a trusted launch of a VM instance **available with the IaaS provider**. A specific requirement is that the trustworthiness of the virtualization environment where the VM instance is launched should be verifiable through an automatic, scalable and least-intrusive way. In the assumed scenario, the client should be able to automatically verify that the launch of the VM image has been performed in a trustable environment.

An additional requirement is that the solution should be implementable using an open source cloud computing platform and should minimize the potential for introducing new vulnerabilities through the implementation of the solution.

3.2.2 Solution requirements

Based on the above defined security aspects of IaaS in public clouds and stated use case, we revisit the requirements for a satisfactory solution to the above defined problem:

- R1: *The launch should provide to a user the mechanisms to ensure that the VM has been launched on a trustworthy host.* In order to establish whether the VM instance launched in the public cloud can be trusted, the client needs to have a verification mechanism to ensure that the VM instance is running on a host which is considered "secure", at least from the software point of view. The verification should be provided by a party or component which is trusted by the client.

- R2: *the client should have the possibility to reliably determine that it is communicating the the generic VM launched on a secure host, and not with a different generic VM instance.* Given that a *generic* VM instance can not, by definition, posses any properties known to the client that would make it identifiable for the client, it is important to provide reliable tools for the CS client to distinguish a trusted VM instance from other types of generic VM instances.

- R3: *The integrity of the VM must be verifiable by the target node* Besides the need to ensure the integrity of the host where the VM instance is run, it is equally important in the scenario of an untrusted cloud service provider to verify the integrity of the VM image. This thesis considers the trusted launch of VMs using generic virtual machines images, i.e. VM images that have not undergone modifications of any kind, something which facilitates verification of the VM images at the time of their launch.

- R4: *Users should have a clear view of the secure launch procedures*, in case the IaaS has certain preferences regarding the software that may or should run on the host where the VM instance is launched. Creating such a capability could contribute to challenging the current perception of lack of transparency, as pointed out in [59]. Furthermore, NIST guidelines name visibility and transparency of the cloud provider processes and mechanisms is one of the criteria for establishing trust in a cloud provider [24].

- R5: *The mechanism supporting the trusted VM launch should be scalable and have a minimum impact on the performance of the cloud computing platform supporting the IaaS infrastructure.* This requirement, which actually consists of two distinct parts is essential for the potential of a designed solution to be implemented in practice. Given the growing scale of cloud computing

[2]In this context, "vendor" is the original entity providing the operating system, e.g. Red Hat for RHEL, or Canonical for Ubuntu, etc.

adoption and the increasing number of hosts employed by cloud providers, any solution with a significant performance hit is likely to have very low adoption. Therefore, while scalability of specific components is out of the scope of this paper, a potential solution should ideally not introduce known bottlenecks that would prevent its adoption for large IaaS deployments.

3.3 Contribution

In the following chapters we examine a scalable solution for secure VM launch and integrity checking in public clouds, to enable trusted launch of generic virtual machine images in trusted clouds. The contribution of this study is both a theoretical description of a *generic* trusted VM launch and an image integrity verification (LIIV) protocol and a description of an implementation design and specific adaptation of the VM LIIV in the scope of an implementation design.

3.3.1 Theoretical contribution

The first part of the theoretical contribution of this study is a transversal overview of the state of the art in cloud computing security, from the web interface of known cloud service providers to the issues on the virtual machine manager and hardware-level vulnerabilities of trusted platform modules.

The second and perhaps more important theoretical contribution of this study is a protocol for generic trusted VM launch on public IaaS platforms. The protocol adopts an abstracted view of cloud computing platform architecture and is aimed to be platform independent. Application of the protocol allows a client to launch a generic VM instance on a public IaaS platform given a certain security profile to verify the integrity of the VM image, as well as ensure that the VM instance has been launched on a host corresponding to the selected security profile. Finally, the protocol provides a way to verify that the client is communicating namely with the VM instance running on the trusted host and not on a different generic VM instance.

3.3.2 Practical contribution

The generic VM LIIV protocol mentioned above and described in full detail in Chapter 5 has been implemented using commodity hardware and OpenStack, an open source cloud management software. Along with the validation of the protocol itself, the implementation offers an insight into the modifications to the OpenStack codebase required in order to implement support for trusted VM launch and integration with the TPM hardware.

Chapter 4
Methodological Approach

4.1 Sources

In order to address both the requirements for academic correctness and industry relevance, the study has used both peer-reviewed articles from collections such as ACM and IEEE, as well as scientific publishers such as Springer and official web resources of the tools and platforms used in the implementation phase of the study.

Given the rapid pace of development and evolution of the OpenStack codebase, information about the state of the art as well as future devlopment plans of the cloud computing platform have been almost exclusively obtained from on-line, non-academic resources. However, even such resources were selected with a critical view and most of the used data is based on the facts obtained from the OpenStack project homepage.

Other information sources, such as blogs, on-line magazine articles and whitepapers were used to get an overall understanding of the field but were not referenced in the thesis.

4.2 Information collection

Following the information collection stage, the goals of the thesis have evolved throughout the project, being influenced by the serendipitous discoveries made during the subject exploration and problem identification stages. This study adapts a *constructive research* approach, which suites the structured but unpredictable nature of research in the information systems field.

As pointed out by Crnkovic in [60], constructive research implies building an artifact with theoretical or practical (and in some cases both) aspects. Such an artifact addresses a domain-specific problem and creates new knowledge and is meant to contribute to the solution of the targeted issue. The current thesis addresses a domain-specific problem introduced in chapter 2 and specifically outlined in chapter 3 and explores theoretical and practical solutions to address the problem. The relevance of the problem is advocated by the overview of the state of the art domain publications based on industry reports and academic research. Considering the contributions described in chapter 3, the current thesis aims to fulfill the criteria for an artifact addressing both theretical and practical research aspects.

A further characteristic of constructive research, according to [60] is that a work adopting a constructive research methodology should yield results which are relevant both theoretically and practically. In addition, such research should address complementary knowledge problems related to feasibility, improvement and novelty. The relevance of the results is ensured by the focus on two narrowed-down propositions defined in chapter 3, framed by the set of five requirements towards the proposed solution, formulated in the same chapter. This set of requirements also addresses the issues regarding feasibility, improvement and novelty of the proposed solution.

Having settled on a constructive research approach, information collection throughout the thesis was shaped by the following stages:

- **Domain exploration** was the first step of the study, required to get a general understanding of the fields related to the thesis. As mentioned above, these fields include *cloud computing*, *virtualization technology* and *trusted computing*. At this stage, information was primarily collected from standards, specifications (as is the case for the Trusted Platform Module specification [29]), guidelines of standardization bodies, such as Definition of Cloud Computing from the National Institute of Standards of Technology (NIST), [17] and Guidelines on Security and Privacy in Public Cloud Computing [24]. The use of standard specifications and documentation directed the study towards a correct use of operational terminology used in the area of cloud computing virtualization technology and trusted computing, as well as an understanding of the interplay between the three areas.

  Among other sources, the current thesis heavily relies on the above referred TPM specification describing the architecture and implementation details of the trusted platform module and the NIST guidelines. However, both are non-academic sources which have not undergone a peer-review process, something which calls for a justification of their use.

  In the case of the TPM specification [29], it has been used only as a source of factual infromation regarding the actual architectural structure and implementation design of the trusted platform module. The document does not contain and has not been used to cite inferred or deducted statements. The fact that [29] it is part the specification provided by trusted computing group for hardware manufacturers provides in the opinion of the author sufficient credibility regarding its relevance and accuracy in describing the structure and capabilities of the trusted platform module.

  When it comes to the referenced documents from NIST, namely [24, 17], they indeed contain subjective information regarding the percieved security risks of cloud computing based on the architecture of cloud computing implementations and deployement models known to the authors. However, despite the fact that the above sources have not been published in a peer-reviewed source, the authors thoroughly reference their work and base it in most cases on peer reviewed sources, making their conclusions both verifyiable and traceable. In addition, the sources authored by the National Institute of Science and Technology which were used in this thesis were selected with a careful and critical approach.

- The aim of the **problem identification** stage was to build an overview of the state of the art in the area of cloud computing and trusted computing as well as obtain an understanding of how the topic of the thesis, namely trusted computing in public cloud computing environments, integrates into the array of research challenges within cloud computing security. This was achieved through a review of the latest and most influential [1] publications with the keywords “cloud computing”, “cloud security”, “hypervisor security”, “trusted computing”, “‘trusted platform module”, “VM manager security”, “IaaS security”, etc. Furthermore, a selected set of references were followed in order to verify the claims of the authors and obtain a deeper understanding of the subject.

- Once the problem was identified and formulated, a second narrowed-down review of the peer reviewed academic publications followed. At this stage, the keywords search was focused on terms as “trusted VM launch”, “‘trusted clouds”, “secure VM launch”, “secure VM migration”. The search was performed in all publicly known peer-reviewed publication databases by using of Google Scholar [2].

[1]Influential publications were considered the ones with the most references

[2]Google Scholar: http://scholar.google.com

- Once a theoretical solution has been formulated, it has to be validated through implementation and integration with one of the open source cloud computing platforms. In this case, OpenStack was chosen based on the criteria specified in chapter 2. At this stage, technical information and community documentation available on the homepage of the project [3] was used for the setup, configuration and maintenance of the OpenStack installation, as well as in order to obtain an understanding of the OpenStack architecture.

- Finally, the generic protocol as well as the prototype implementation were reviewed based on the requirements and problem statement defined in chapter 3.

[3]The OpenStack project http://openstack.org

CHAPTER 5

Secure VM Launch and Migration Protocol

This chapter introduces a platform-agnostic secure launch protocol for a generic virtual machine image (GVMI). Generic virtual machine images are virtual machine images that do not differ from the vendor-supplied VM images (colloquially known as "vanilla software") [1]). They are made available by the IaaS providers for clients that intend to use an instance of a VM image that was not subject to any modifications, such patches or injected software. The protocol described in this chapter allows a client that requests a GVMI to ensure that it is run on a trusted platform. The concept of GVMI is also explained in further details below.

5.1 Attacker model

The use cases for a trusted VM launch in public clouds assumes that several parties are involved, such as the following:

5.1.1 Malicious IaaS provider

In the context of the proposed protocol, the domain of the *IaaS provider* is generally considered to be untrusted. That includes the deployment of the cloud management platform, as well as the hardware and software configuration of the physical machines supporting the IaaS infrastructure. The untrusted domain also includes the communication between servers that are part of the IaaS platform, as well as the generic VMs made available by the IaaS provider (although it is assumed that they are identical as the ones supplied by the vendor).

However, this attacker model considers that the *physical security* of the hardware and the integrity of the TPM is ensured. This is important in order to be able to rely on the security model of the Trusted Computing Group (TCG), since TCG's model is not designed to withstand physical attacks [28]. This assumption builds on the fact that the TPM is tamper-evident [61] and a visual inspection would be enough to discover a hardware attack.

[1]In this context, "vendor" is the original entity providing the operating system

5.1.2 Other actors

The *client* is a user of cloud computing services and intends to launch or use a VM. The client can be both technically skilled (e.g. capable to assessing the security of platform configurations based on values from the measurement list, etc.) and a non-expert that requires access to a generic VM instance launched and running on a trusted platform.

The *Trusted third party (TTP)* is, as the name implies, trusted by both the *Client* and the *Cloud service provider*. The breaches of Certificate Authorities during 2011 have emphasized the drawbacks of centralized security models and their susceptibility to attacks [62]. The more complex the operations performed by the TTP, the higher the probability of it having exploitable vulnerabilities. It is therefore important to keep the implementation of the TTP as simple as possible. The main task of the TTP is to attest the configuration of the nodes that will host the generic VMs and asses their security profile according to some predefined policies. Within the current trust model, TTPs could be implemented on the client side, as long as the IaaS provider agrees to that and the client has the capability to set up and operate an attestation and evaluation engine.

5.1.3 On generic virtual machine images

A peculiar aspect of generic virtual machine images is that they by definition can not posses any verifiable properties that could distinguish two different instances launched using a GVMI. That is, all of the GVMI of a particular distribution offered by the vendor are *binary identical*.

This property of GVMI makes it difficult for a IaaS client to verify that the virtual machine instance it interacts with runs on a particular hardware or software stack, since as mentioned above, the VM instance launched from a GVMI does not possess any unique properties.

In the case of trusted launch of a generic VM, it is essential for the client to be able to ensure both the integrity of the underlying platform and of the VM image supplied by the IaaS provider. The fact that all GVMI are identical can be used in the context of a secure launch protocol in order to verify that a generic VM image has been launched on a *trusted* platform.

5.1.4 Specific attacker model

The situation when a non-expert user requires the launch of a VM on a secure platform implies a recommendation that such VMs should generally not to be used for business-critical operations. However, since this generic VM will be part of the security perimeter of a larger organization, it is important to provide a security level that is as high as the setup allows. Hence, the following attacker actions are likely in this situation:

- The IaaS provider ignores the request for launching the VM on a *trusted* platform and launches the VM on a generic platform. This situation is addressed by requirement R1 and R4.
- The IaaS provider launches *a VM* on a trusted platform, but alters the generic VM (e.g. by injecting forged SSL certificates) in order to intercept the communication between the client and the VM to obtain valuable information (addressed by requirement R3).

Revisiting requirement R2, in the following trusted launch protocol, obtaining a correct response to a challenge from the client to the VM (the object of the challenge being a secret nonce which is sealed by the *TTP* on the destination node after it has been attested) is a **sufficient proof** that the VM is launched on a trusted platform.

5.2 A secure launch protocol for generic VMs

This section describes a secure launch protocol based on the assumptions and limitations above. The protocol is designed to be implementable on any open source cloud management platforms and does not employ any platform-specific considerations.

5.2.1 Platform-agnostic protocol description

The following steps are required in order to perform a trusted generic VM launch.

- Before initiating the launch procedure, client $\mathcal{C}$ generates an 1024-bit long nonce denoted as $\mathcal{N}'$ **(1)**, which will be used as a proof token in communications between the client and the VM and must be kept secret throughout the launch process, as shown in Figure 5.1.
- Next, $\mathcal{C}$ creates a token $\mathcal{T}$, containing $\mathcal{N}'$, the preferred security profile ($\mathcal{SP}$) and the hash of the VM image type that is to be launched (H_{VM}). The token is encrypted with the public key of TTP, noted as $\mathcal{T}_{PK'}$ To improve user experience these actions could be performed transparently to the user by a web browser plugin when navigating to the cloud control web interface. **(2)**.
- Further, $\mathcal{C}$ requests *cloud controller (CC)* to load a generic VM by providing the following parameters in the request **(3)**:
  - VM type (e.g. Ubuntu 12.04)
  - Required security profile
  - URL of the TTP
  - Token $\mathcal{T}_{PK'}$ generated in step **(2)**

  The security profile will determine the lower bound of trust level that is required from the host $\mathcal{H}$ on which the VM will run, with stricter security profiles accepted.
- In the next step, *CC* schedules a VM on the appropriate node, depending on its membership in the respective security profile group **(4)** and sends a request to generate a bind key PK^{Bind}, also providing the URL of the TTP.
- Once the destination host $\mathcal{H}$ receives the bind key request, it retrieves the PCR-locked non-migratable TPM-based bind key PK^{Bind}. This key can be periodically regenerated by $\mathcal{H}$ according to a administrator-defined policy, using the current platform state represented by the TPM PCR. It is important to note that the values of the PCRs should not necessarily be in a trusted state in order to create a trusted state bind key **(5)**
- Next, $\mathcal{H}$ retrieves the TPM_CERTIFY_INFO structure by calling the `TPM_CERTIFY_KEY` TPM command, where the structure of TPM_CERTIFY_INFO consists of a hash of the bind key PK^{Bind} and the hash of the PCR values used to create PK^{Bind}, denoted as $\{H_{PK^{Bind}}, H_{PCR\_INFO}\}$ **(6)**.
- $\mathcal{H}$ sends an attestation request to the TTP using the URL initially supplied by the client. The arguments sent with the request to the TTP are represented as follows:
  - Client-provided token $\mathcal{T}_{PK'}$
  - *Attestation data*, which includes the public bind key, the TPM_CERTIFY_INFO structure, the hash of TPM_CERTIFY_INFO signed with the *Attestation Identity Key (AIK)* [2], the *Integrity Measurement List (IML)* and the *AIK certificate* followed by a session nonce, collectively represented as: $\{PK^{Bind}, TPM\_CERTIFY\_INFO, {H_{TPM\_CERTIFY\_INFO}}^{AIK}, IML, AIK-cert, \mathcal{N}^{session}\}$ **(7)**.

[2]Expressed as ${H_{TPM\_CERTIFY\_INFO}}^{AIK}$

- TTP uses its private key PrK', which corresponds to the public PK' to attempt to decrypt the token $\mathcal{T}_{PK'}$ **(8)**.
- TTP validates the attestation information received from $\mathcal{H}$ through the following actions **(9)**:
  - Validates the structure $TPM\_CERTIFY\_INFO$
  - Validates the key PK^{Bind}
  - Calculates the hash of the PCR values H_{PCR} based on the information in the *IML* and compares it with the digest of PCR_INFO, which is a component of TPM_CERTIFY_INFO
- TTP examines the entries in the IML in order to determine the trustworthiness of the platform [3] and decides whether the security preference $\mathcal{SP}$ is satisfied by the current configuration of node $\mathcal{H}$ **(10)**.
- If that is true TTP encrypts the nonce $\mathcal{N}'$ and the hash H_{VM} with the bind key PK^{Bind} obtained from $\mathcal{H}$, in order to ensure that the secure token $\mathcal{N}'$ is only available to $\mathcal{H}$ in a trusted state **(11)**. Through the act of sending $\mathcal{N}'$ encrypted with the public key PK^{Bind} available to the trusted configuration of $\mathcal{H}$, the security perimeter expands to include three parties: $\mathcal{C}$ itself, stateless TTP and node $\mathcal{H}$ in its trusted configuration. This has the implications that all actions performed by $\mathcal{H}$ in its trusted configuration are trusted by default.
- Prior to launching the VM, node $\mathcal{H}$ decrypts $\mathcal{N}'$ using the TPM-issued PrK^{Bind}, which is available to it in its trusted configuration but stored in the TPM; next $\mathcal{H}$ compares H_{VM} obtained from the TTP with the hash of the provided VM image and accepts the image for launch only in case the values are equal (**12**).
- Finally, $\mathcal{H}$ **(13)** injects $\mathcal{N}'$ into the VM image prior to launching the VM.
- To confirm a successful launch, $\mathcal{H}$ returns an acknowledgement to CC **(14)**.
- To verify that the requested VM image has been launched on a secure platform, $\mathcal{C}$ challenges the VM launched on host $\mathcal{H}$ to prove its knowledge of $\mathcal{N}'$. Since $\mathcal{N}'$ will become known to TTP, it should not be used as an encryption key. However, in the case when the TTP is implemented and operated by $\mathcal{C}$, $\mathcal{N}'$ could be used as a key to e.g. establish a secure communication channel (such as an IPSec tunnel) between $\mathcal{C}$ and the *VM* running on $\mathcal{H}$ **(15)**

5.2.2 Security analysis

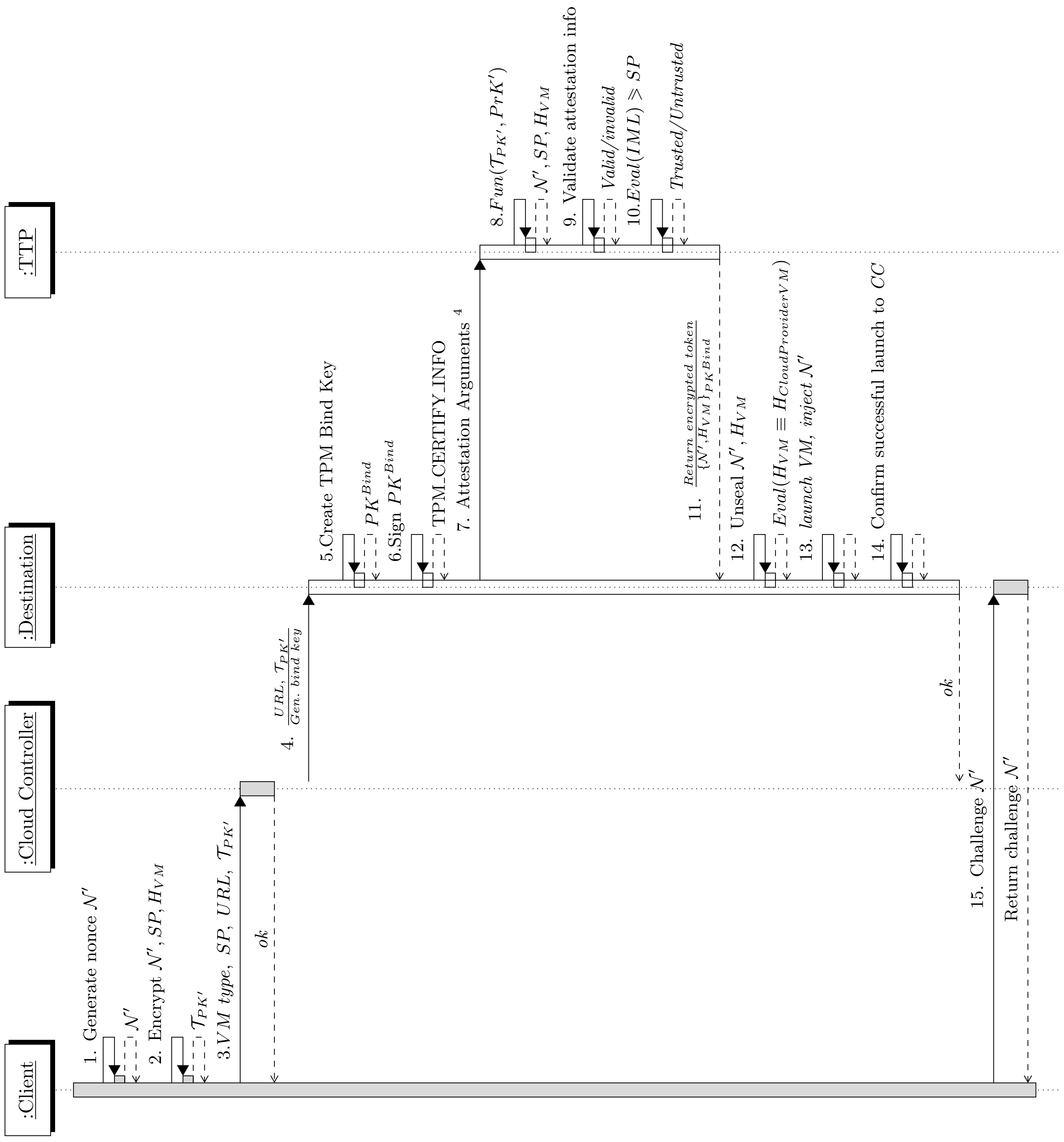
As a result of the above protocol, the client $\mathcal{C}$ and the launched guest VM instance on node $\mathcal{H}$ have a shared secret $\mathcal{N}'$. $\mathcal{C}$ can then challenge its VM residing on $\mathcal{H}$ to check the knowledge of $\mathcal{N}'$. Returning to the security concerns of $\mathcal{C}$, expressed in the requirements towards the trusted launch protocol formulated in chapter 3, they are addressed as follows:

- R1: The fact that a VM is running on a *trusted platform* is ensured by the properties of the bind key used to seal the shared secret $\mathcal{N}'$ to the trusted configuration of host $\mathcal{H}$;
- R2: The fact that $\mathcal{C}$ is communicating with *the VM launched on a trusted platform (and not a different generic VM running on an untrusted platform)* is ensured by the possession of a secret token $\mathcal{N}'$ encrypted with $\mathcal{H}$'s PCR trusted configuration-bound TPM key and only available when

[3] Aspects of evaluating what actually *is* a trusted platform are out of the scope of the protocol

[4] $\mathcal{T}_{PK'}, PK^{Bind}, TPM\_CERTIFY\_INFO, H_{TPM\_CERTIFY\_INFO}{}^{AIK},\ IML, AIK-cert, N^{Session}\}$

Figure 5.1: Trusted generic VM launch protocol


:Client
:Cloud Controller
:Destination
:TTP
1. Generate nonce $\mathcal{N}'$
$\mathcal{N}'$
2. Encrypt $\mathcal{N}', SP, H_{VM}$
$\mathcal{T}_{PK'}$
3.VM $type$, SP, URL, $\mathcal{T}_{PK'}$
ok
4. $\frac{URL,\ \mathcal{T}_{PK'}}{Gen.\ bind\ key}$
5.Create TPM Bind Key
PK^{Bind}
6.Sign PK^{Bind}
TPM_CERTIFY_INFO
7. Attestation Arguments [4]
8.$Fun(\mathcal{T}_{PK'}, PrK')$
$\mathcal{N}', SP, H_{VM}$
9. Validate attestation info
$Valid/invalid$
10.$Eval(IML) \geqslant SP$
$Trusted/Untrusted$
11. $\frac{Return\ encrypted\ token}{\{\mathcal{N}', H_{VM}\}_{PK^{Bind}}}$
12. Unseal $\mathcal{N}', H_{VM}$
$Eval(H_{VM} \equiv H_{CloudProviderVM})$
13. $launch$ VM, $inject$ $\mathcal{N}'$
14. Confirm successful launch to CC
ok
15. Challenge $\mathcal{N}'$
Return challenge $\mathcal{N}'$

$\mathcal{H}$ it is in a certain configuration considered 'trusted'. Considering the fact that a change in the software stack of $\mathcal{H}$ would make $\mathcal{N}'$ unavailable, $\mathcal{C}$ has a certain guarantee that the VM possessing $\mathcal{N}'$ is running on a trusted platform.

- R3: Integrity of the VM image is ensured through the verification performed by node $\mathcal{H}$, which compares the VM image to be launched with the "expected" hash H_{VM}, provided by the client. $\mathcal{H}$ must be running a trusted configuration at the time when it retrieves the generic VM image provided by the cloud platform through the image store in order to access the reference hash H_{VM}.

- R4: Transparency of the trusted VM launch procedure is ensured by the introduction of client parameters, such as the URL of the TTP, the trust level of the VM host and the secret token generated by the client. The ability to choose the TTP opens the possibility for the client to ensure the trustworthiness of the host attestation procedure, either through audit controls of the TTP or by itself serving the role of the TTP (in case the cloud service provider agrees to that).

- R5: While the actual performance of the protocol depends on the specific implementation and must be verified in a realistic setting, the protocol does not display any elements that are, at least at this stage, known to not be scalable.

Regarding the security of the client-generated secret $\mathcal{N}'$, it is worth noting that throughout the course of the protocol, $\mathcal{N}'$ has only been available in cleartext to $\mathcal{C}$, (which generated it), TTP which has sealed it to $\mathcal{H}$ and finally $\mathcal{H}$ *once considered to be in a trusted state.*

An additional advantage is the stateless nature of the TTP, which implies that it does not maintain knowledge of $\mathcal{N}'$ except for the moment of sealing it to $\mathcal{H}$. As a result, an attacker can only obtain $\mathcal{N}'$ from TTP if they obtain TTP's private key PrK'. However, it is assumed that TTP ensures the confidentiality of its private key. Furthermore, assessment of a hosts' trust level according to a deterministic algorithms which only takes one two inputs (in the form of static set of reference measurement data and dynamic attestation calls from any $\mathcal{H}$) will be easily traceable and reproducible based on the original input data, without the need to recreate or rely on a certain state of the TPP's internal data. Finally, a stateless architecture of the TTP contributes indirectly towards requirement R5.

By maintaining a minimalistic, transparent structure that relies only on the secrets created by the client, the TTP and the TPM, the protocol corresponds to Kerckhoff's principle [63], according to which the security of a cryptosystem must not depend upon keeping the crypto algorithm secret, rather only depends on keeping the key secret. In order to further address requirement R4, all of the parties involved in the attestation process could log transactional information to inform the client about the progress of the trusted launch procedure. However, such functionality is not addressed in this thesis.

5.2.3 Enhancement areas

A potential *vulnerability* that requires attention is the post-launch modification of $\mathcal{H}$'s software stack. The runtime process infection method, which is a method for infecting binaries during runtime [5] is one of the malicious approaches that could be used in this situation. This scenario is in fact a common threat to all TCG-based systems, also touched upon in [64]. A related attack strategy is described in detail in [65]. However, such attacks are a common threat to all TCG-based systems and should be prevented using means within the platform which is part of the trusted computing base verified at boot time, the presence of which is verified by the above protocol.

From a *client perspective*, the secure launch protocol can be improved by reducing the number of steps that need to be performed prior to initiating the VM image launch. That would make implementations of the protocol more user-friendly and reduce the implementation efforts on the client side. However, we consider that the architecture of the current protocol does not contain any design decisions that make it

[5] Runtime process infection, http://www.phrack.org/issues.html?issue=59&id=8&mode=txt

impossible to further reduce during implementation stage the set of actions that need to be performed by the user.

CHAPTER 6
Implementation Design

6.1 Implementation model

The trusted virtual machine launch protocol described in Chapter 5 was implemented using commodity hardware and software in order to practically verify the protocol's implementability and performance.

Several components are essential for the implementation of the above protocol, namely a cloud computing platform deployed on one or more hosts with at least one hardware TPM chip per compute host and network communication between the hosts. In the current implementation we used two physical hosts, where one host ran an OpenStack Compute service and the other host ran the other required OpenStack services as well as the "Trusted Third Party" service. Communication between the nodes was established through a routed Ethernet connection over a Cat6 cable.

6.1.1 Controller node setup

The cloud controller was hosted on a Dell OptiPlex 170L with a Intel(R) Pentium(R) 4 CPU 2.80GHz processor and 1 GB memory. No special hardware support was required, so a generic version of Ubuntu (Precise) 12.04 was installed. *Devstack* [1] was used without any major modifications in order to install the *nova-compute, nova-cert, nova-volume, nova-scheduler, nova-consoleauth* and *nova-network* services on the controller node.

6.1.2 Compute node setup

The compute node was hosted a Dell PowerEdge 310 with a Intel(R) Xeon(R) CPUX3450, 2.67GHz and 8 GB memory. The host was equipped with a TPM chip 1.2 Level 2 Revision 116 model ST33TPM12LPC from STMicroelectronics [2] and was installed with Ubuntu (Lucid) 10.04 LTS which was subsequently upgraded to Ubuntu (Precise) 12.04 LTS. In order to enable support for TPM and integrity measurements, Trusted GRUB and Linux-IMA were additionally configured on the host.

In order to enhance the standard GRUB into a version that offers TCG support, the Trusted GRUB patch [3] for GRUB version 0.97 was installed. Generic Ubuntu 10.04 is shipped with GRUB version 1.98, so GRUB was downgraded to version 0.97.

The Linux Integrity Subsystem, implemented with Linux-IMA in kernel version 2.6.30 provides several integrity functions, namely: *collect, store, attest, appraise and protect.* Ubuntu 10.04 is shipped with the

[1] The development OpenStack environment installation script, http://devstack.org/

[2] Product specification at http://www.st.com/internet/mcu/product/252378.jsp

[3] TrustedGRUB project page on Sourceforge http://sourceforge.net/projects/trustedgrub/

kernel version 2.6.32-25 which includes the IMA modules but does not have the IMA enabled by default. Thus, the kernel was recompiled with the option `CONFIG_IMA` in order to collect the runtime parameters measured by the TPM. The TPM software stack deployed on the compute node is visualized in figure 6.1

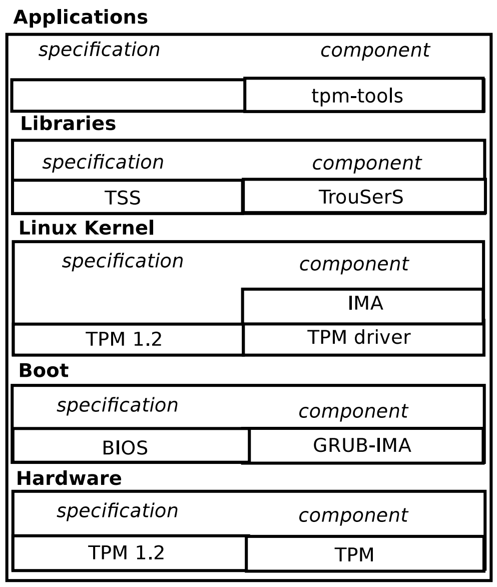


Figure 6.1: TPM stack deployed on the compute node

Devstack currently supports Ubuntu distributions 11.10 and 12.04, but a set of modifications was required in order to enable the multinode install of OpenStack. The devstack installation script was configured to install the *nova-compute* and *nova-volume* services on the compute host.

6.2 OpenStack

6.2.1 OpenStack API

OpenStack supports two user API interfaces, namely OpenStack API (currently at version 1.1) and the EC2 API[4], the latter being an open source implementation of the Amazon Elastic Compute Cloud API. A special Admin API is available in order to perform administration operations by privileged users.

According to the notes from the OpenStack summit in December 2011, the EC2 API will eventually be deprecated [5]; furthermore, the OpenStack API is being exclusively developed to reflect the OpenStack architecture and feature set. This makes the OpenStack API a more appropriate candidate to be used during the implementation of a trusted VM launch protocol.

[4]http://nova.openstack.org/devref/api.html
[5]http://www.openstack.org/blog/tag/diablo/

6.2.2 Implementation considerations

The fith release of OpenStack Nova (”Essex”) was used for implementation of the above protocol. OpenStack Nova “Essex” was released on April, 5th, 2012.

Throughout the protocol implementation, the following principles had prevailing importance for the implementation design:

- Modifications to the underlying codebase (which consists of OpenStack Nova, OpenStack Horizon and a Python API client) were kept down to a minimum
- The general encapsulation (here in the sense of transparency of operation related) principles observed in OpenStack Nova should be maintained. This implies maintaining separation of concerns between the OpenStack components, specifically relevant in case of implementing support for remote attestation and key sealing functionality on the compute nodes. Additional functionality should not add unrelated functionality to modules other than the ones directly affected by the functionality.
- Implementation of the trusted launch protocol should have minimal or ignorable effects on the performance on the system as a whole (i.e. both in the case of trusted launch and standard operating mode).
- Implementation-specific deviations from the above trusted launch protocol should not break the trust chain described in the protocol.

6.3 Implementation design description

By and large, the generic VM launch protocol does not require radical modifications to OpenStack's codebase for implementation. Before a description of the proposed modifications to the codebase, several issues must be noted. First, the **asynchronous, message-based** architecture of OpenStack is essential for its scalability. Hence, in the process of launching a VM, all communication implemented as an *RPC cast* (typically until the compute is assigned and takes responsibility for the launch of the instance) should be maintained as such. Along with implications for the implementation design, this results in that a `run_instances` call will return an acknowledgement from the scheduler after it casts the message to launch the instance on a selected compute node, before the actual VM is started. In case the VM launch will fail, the results will be displayed on the dashboard.

Second, considering the above description of the available API implementations, the OpenStack API (rather than the EC2 version) will be used throughout the implementation.

Third, in order to limit the performance hit of node attestation as much as possible, as well as encapsulate the tasks performed by respective components, it is suggested that the attestation procedure is done after the compute node has verified whether the instance is not already running and prior to the launch of the VM on the host.

Fourth, detection of specific security vulnerabilities in the software stack of the compute host is out of the scope of this paper. Rather, the aim is to collect and provide dependable configuration information to an integrity appraisal party. It is assumed that such detailed information about the software stack of a host is sufficient to assess whether the respective host can be included in the security perimeter of the client. Thus, evaluation of the host's integrity is done by recalculating the hashes reported in the `binary_bios_measurements` and `binary_runtime_measurements` and comparing to respectively the `boot_aggregate` entry and the value of `PCR10` .

6.4 Proposed Trusted VM Launch Protocol Implementation

In order to ensure that the study can be replicated, the maximum amount of details about the implementation have been provided below.

A simple message sequence diagram for the implementation design is provided in figure 6.2

6.4.1 OpenStack implementation model

The implementation of the protocol in OpenStack requires changes on several levels of the platform. They are described in order from the user interface (dashboard) to the interface communicating with the `TPM Interface (TPMI)` middleware component.

Horizon

The Horizon dashboard has been modified in order to accept additional input from the user, namely an `Attest host` choice **(1)** for the user to select a trusted launch procedure, as well as a drop-down list for the `Minimum accepted security profile` on a scale of 1 to 10 **(2)**. In addition, an input field is available for the base64 encoded encrypted token generated by the client and denoted as $\mathcal{T}_{PK'}$ **(3)** and the URL of the preferred TTP **(4)**. The input method for the client-generated token is in itself less important and affects primarily the usability of the solution. Alternative solutions, such as background daemons and browser plugins can be used to facilitate the trusted launch procedure. Server-side generation of client token is however not possible since in that case the cloud service provider would have the knowledge of the internals of the token during the token generation phase.

Nova API

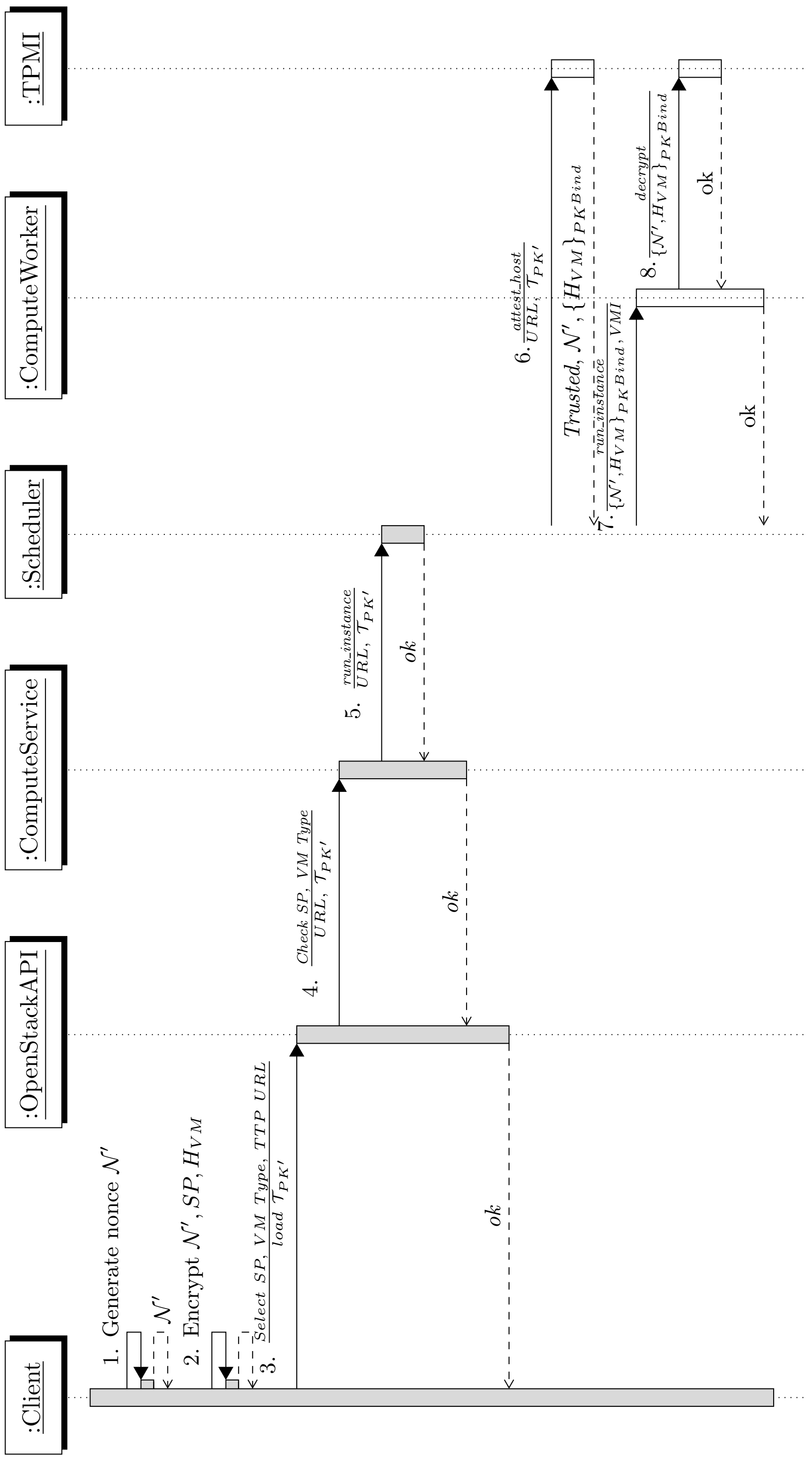
The API interface of the Nova component required insignificant modifications through several modules in order to forward the client-generated token $\mathcal{T}_{PK'}$, the attestation preference and the trust level preference from the client to the scheduler. The changes in Nova API can also be reused by other existing frontends (e.g. the OpenStack command-line API or the Amazon EC2 API, that will however require certain modifications not covered in this thesis).

Scheduler

OpenStack Nova features a plugin scheduler architecture, which allows for simple modification of the scheduling mechanism as well as development of new schedulers. The schedulers available in the current “Essex” OpenStack edition are `simple, chance, distributed_scheduler, multi` and `vsa`. The `simple` scheduler has been chosen for modification in this implementation.

- According to normal behavior, the scheduler produces a list of eligible hosts to run the virtual machine. At this point, the trusted launch client choice is verified by examining **(1)**.
- In case the client has requested a trusted VM launch, the scheduler performs a DB lookup to find a host with a security profile which is larger or equal than **(2)**. OpenStack currently holds tables with per-host information, rather than per-tenant information, since per-tenant information would require integration with Keystone. The host security profile information will be pre-stored in the DB by the cloud service provider and made accessible to the trusted launch process. Failure to do so would effectively mean a denial of service, something which is not in the interest of the cloud service provider in this scenario’s attack model.
- If the host, according to the information stored in the DB has a security profile which satisfies the requirements of the client, the scheduler sends an RPC call containing elements **(3)** and **(4)** to the host (i.e. to the `nova-compute` process of the host) to perform an attestation. If the security

Figure 6.2: Trusted generic VM launch implementation design


:Client
:OpenStackAPI
:ComputeService
:Scheduler
:ComputeWorker
:TPMI
1. Generate nonce N'
N'
2. Encrypt N', SP, H_{VM}
3. $\frac{Select\ SP,\ VM\ Type,\ TTP\ URL}{load\ \mathcal{T}_{PK'}}$
ok
4. $\frac{Check\ SP,\ VM\ Type}{URL,\ \mathcal{T}_{PK'}}$
ok
5. $\frac{run\_instance}{URL,\ \mathcal{T}_{PK'}}$
ok
6. $\frac{attest\_host}{URL;\ \mathcal{T}_{PK'}}$
$Trusted, N', \{H_{VM}\}_{PK^{Bind}}$
7. $\frac{run\_instance}{\{N', H_{VM}\}_{PK^{Bind}}, VMI}$
ok
8. $\frac{decrypt}{\{N', H_{VM}\}_{PK^{Bind}}}$
ok

profile of the host does not satisfy the client requirements, the scheduler will iterate through the host list until a suitable host $\mathcal{H}$ is found.

- `nova-compute` performs an attestation using **(3)** and **(4)** according to the protocol described in 6.4.2 and also in figure 6.3. In case the attestation is successful, nova-compute returns to the scheduler the token received from the TTP, marked as $\{\mathcal{N}', H_{VM}\}_{PK^{Bind}}$.
- Having enough information about the trust level of $\mathcal{H}$ and the encrypted token, the scheduler performs an `RPC cast` to $\mathcal{H}$, in order to launch the requested VM.

Compute

At this point, $\mathcal{H}$ is part of the client's security perimeter and is in a state which allows it to access the PrK^{Bind} stored in the TPM and as a result decrypt the token using `TPMI`.

Appendix C presents the `_run_instance` function in **/compute/manager.py**, which performs a check on the VM instance to ensure that it has not already created and in case of a positive outcome obtains an IP address for the instance and finally spawns the instance. The token decryption step is done after the call `self._check_instance_not_already_created`.

Next, the nova-compute process on $\mathcal{H}$ performs the following steps:

- Applies a hash function on the generic VM image obtained from the Glance component (offered by the cloud service provider)
- Compares the hash with the reference value obtained from the client, H_{VM}
- If the hashes are identical, the client-originated nonce $\mathcal{N}'$ is injected into the generic VM image and the VM is launched.

6.4.2 Detailed Interaction with the TTP

A call from `compute/manager.py` sends a call to a `TPMI` which initiates the communication with a TTP (having knowledge of the TTP URL) and the attestation procedure, as shown in Figure 6.3.

- `TPMI` retrieves a pre-generated TPM keypair PK^{Bind}, PrK^{Bind} with the TSS command `TPM_CREATE_KEY` **(0)**
- `TPMI` retrieves the TPM_CERTIFY_INFO structure, pre-generated with the TSS command `CERTIFY_KEY` Both the keypair PK^{Bind}, PrK^{Bind} and the TPM_CERTIFY_INFO structure can be periodically regenerated by $\mathcal{H}$ according to a administrator-defined policy, using the current platform state represented in the TPM PCR **(1)**
- `TPMI` will generate a nonce and send an attestation request **call** to the TTP **(2)**, where it sends: the session nonce $\mathcal{N}^{Session}$, TPM public key PK^{Bind}, the encrypted token $\mathcal{T}_{PK'}$ and the attestation credentials: TPM public bind key, and the TPM_CERTIFY_INFO construct along with its hash signed with the attestation identity key; the integrity measurement list and the attestation identity key are also sent: PK^{Bind},$TPM\_CERTIFY\_INFO$, $\{H_{TPM\_CERTIFY\_INFO}{}^{AIK},\ IML, AIK - cert\}$
- On the TTP side, once a message from $\mathcal{H}$ is received, `tcp_listener` spawns a `ttp_worker` process **(3)**.
- `ttp_worker` attempts to decrypt the token to verify whether the token was intended for it **(4)**.
- `ttp_worker` validates the attestation arguments (TPM_CERTIFY_INFO, PK^{Bind}, verifies signature of PCR_INFO) **(5)**.

- `ttp_worker` parses the IML to evaluate the trustworthiness of the software stack on the host and assigns a security profile SP'' based on the values of the IML **(6)**
- `ttp_worker` evaluates $SP'' \geqslant SP$ to identify whether the host platform $\mathcal{H}$ is trustable, i.e. fulfills the security profile requirements of the customer **(7)**.
- In case the host is trustable, `ttp_worker` encrypts $\{\mathcal{N}', H_{VM}\}_{PK^{Bind}}$. The size of the token is 512 bits (a 256-bit nonce $\mathcal{N}'$ and the 256 bit long SHA-256 hash of the VM image), which is significantly lower than a minimum RSA keysize of 1024 bits (the current implementation uses an asymmetric encryption key size of 4096 bits) **(8)**.
- `ttp_worker` sends the reply token including the secret nonce obtained from the client and the hash of the virtual machine to be loaded, encrypted with the bind key: $\{\mathcal{N}', H_{VM}\}_{PK^{Bind}}$; the session nonce $N^{Session}$ is also sent for session identification **(9)**.

Once TTP returns an acknowledgement to `TPMI`, it in turn decrypts the token $\{\mathcal{N}', H_{VM}\}_{PK^{Bind}}$ using the TPM-stored private key PrK^{Bind}. Next, the function in `TPMI` sends a call to manager.py that injects the obtained $\mathcal{N}'$ and continues the launch process.

In case the attestation or sealing procedure fails at any point in time, or the call times out due to high load on the TTP, the process in manager.py exits with an exception.

Implementation of TPMI

In order to support the proposed protocol, the `TPMI` exposes the following public functions:

- **attest_host/2** – should be called from manager.py in order to attest the host and obtain the $\mathcal{N}'$, H_{VM}

  ```
  initiate_attestation(URL  ::string(),
                      Token::string(),
                       ) ->  {N' , H_VM} OR {error, Reason}
  ```

  internally, calls initiate_attestation/2.

- **unseal/1** – function to unseal the TPM-key (PK^{Bind}) encrypted token received from TTP as a result of the attestation:

  ```
    unseal(Token::string()) -> {N'::string(), SP::integer(), H_VM::string()}
  ```

  returns the client's secret nonce $\mathcal{N}'$, security profile SP and the hash of the generic VM respectively.

Other functions should not be exposed:

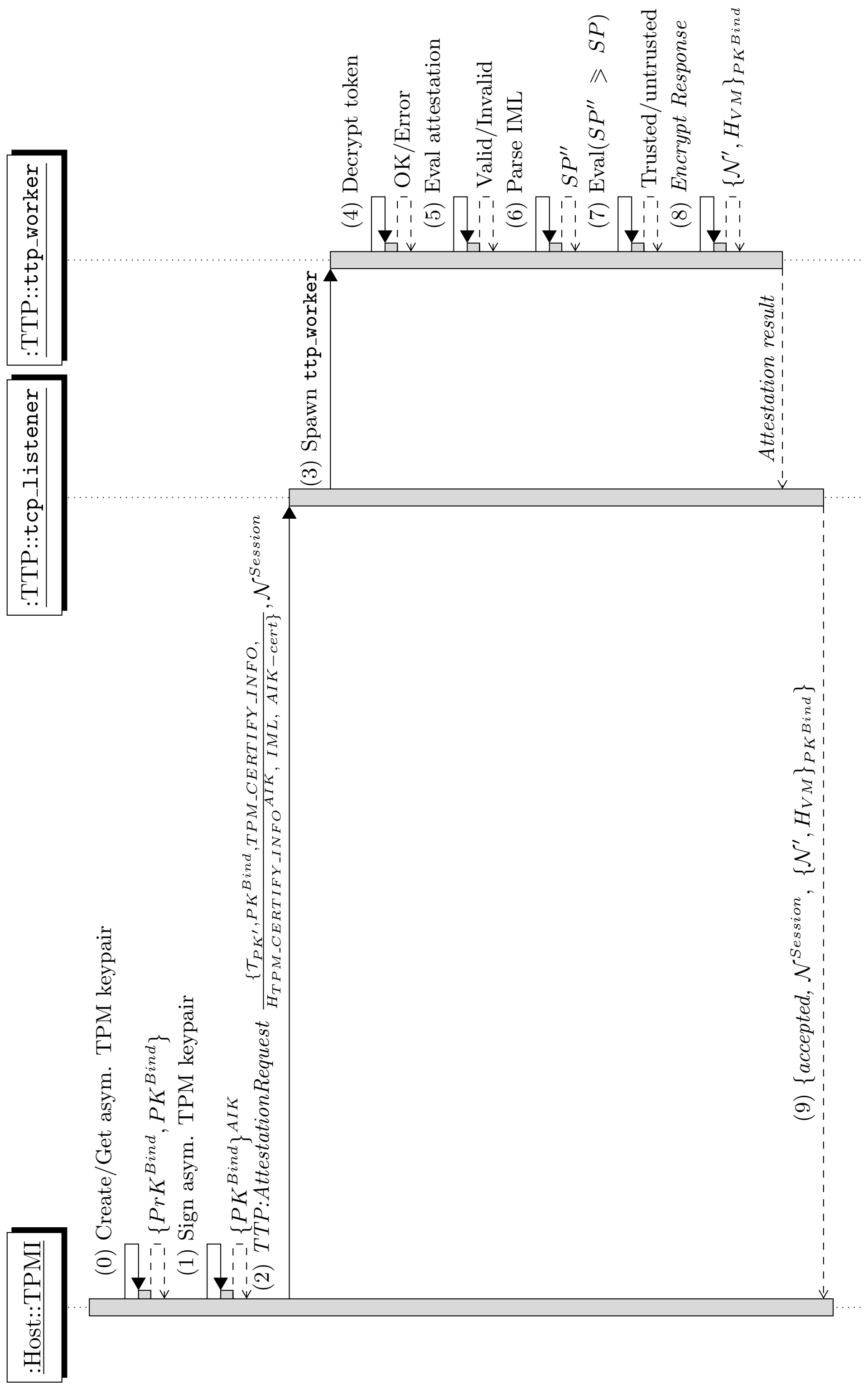
- **initiate_attestation/2** – function to attest the host with a Trusted Third Party prior to a trusted VM launch:

  ```
  initiate_attestation(URL  ::string(),
                       Token::string(),
                       ) -> {accepted, SealedToken} OR {error, Reason}
  ```

  the arguments are respectively:

  1. URL: the url of the TTP, obtained from the client;

Figure 6.3: Host to TTP communication


:Host::TPMI
:TTP::tcp_listener
:TTP::ttp_worker
(0) Create/Get asym. TPM keypair
$\{PrK^{Bind}, PK^{Bind}\}$
(1) Sign asym. TPM keypair
$\{PK^{Bind}\}^{AIK}$
(2) $TTP{:}AttestationRequest\ \frac{\{T_{PK'}, PK^{Bind}, TPM\_CERTIFY\_INFO,}{H_{TPM\_CERTIFY\_INFO}{}^{AIK}, IML, AIK-cert\}}, N^{Session}$
(3) Spawn ttp_worker
(4) Decrypt token
OK/Error
(5) Eval attestation
Valid/Invalid
(6) Parse IML
SP''
(7) Eval($SP'' \geqslant SP$)
Trusted/untrusted
(8) *Encrypt Response*
$\{N', H_{VM}\}_{PK^{Bind}}$
Attestation result
(9) $\{accepted, N^{Session}, \{N', H_{VM}\}_{PK^{Bind}}\}$

2. Token: encrypted token passed from the client, containing the nonce, security profile and hash of the generic VM

returns the tuple **{accepted, SealedToken}** OR **{error, Reason}**, where **SealedToken** is a token containing the client's secret nonce $\mathcal{N}'$ and the hash of the generic VM, all encrypted with the public key PK^{Bind}.

This function can have the following structure (pseudocode):

```
check TPM keychain
    if no keychain available
call gen_tpm_keys/0 to generate TPM keychain
    end
try open_tcp_session_to_TTP(URL)
    send Token, IP to TTP, N, etc.
    receive confirmation of successful attestation
    and token $\{N', H_{VM}\}_{PK\hat{}Bind}$
catch
     tcp_session_failed -> propagate_to_dashboard:ttp_unreachable
catch
     attestation_failed -> propagate_to_scheduler:attestation_failed
catch all -> abandon_launch
after
call unseal/1 to decrypt the token $\{N', H_{VM}\}_{PK\hat{}Bind}$
return $\{N', H_{VM}\}$ | $\{$error, Reason$\}$
```

- **gen_tpm_keys/0** – calls underlying libraries to generate an asymmetric keypair from the TPM (using the TPM command TPM_CREATE_KEY).

  `gen_tpm_keys() -> PK`^{Bind}`::string()`

  returns an asymmetric Public Key, while the private key is kept in the TPM.

- **sign_tpm_keys/1** – signs the public key PK^{Bind} created by the TPM with the Attestation Identity Key, using the CERTIFY_KEY TPM command.

  `sign_tpm_keys(PK`^{Bind}`::string()) -> {`$H_{PK^{Bind}}{}^{AIK}$`::string(), `$H_{LockedPCR}$`::string()}`

  returns a token containing the public key PK^{Bind} signed with the TPM's AIK and a hash of the PCR values to which the asymmetric key is bound.

All of the methods described above, with the exception of *attest_host/2* should be kept private in order to minimize the coupling with other OpenStack modules and keep the internals of `TPMI` easily modifiable.

6.4.3 Implementation of the TTP

A prototype version of the TTP has been implemented for attestation purposes. It consists of the following components:

- **tcp_listener** a *supervised* process that accepts the incoming tcp connections, maintains the session and spawns **ttp_worker** processes to process the attestation requests. For a production-quality implementation, "tcp_listener" should be a scalable server capable of multiple concurrent connections, that is however not needed for the prototype.

- **ttp_worker** is the main process of the TTP. In particular, it has the following responsibilities:
  - decrypt the token $\mathcal{T}_{PK'}$, in order to verify whether the token is addressed to the correct *that* TTP.
  - Perform the validation and attestation based on the arguments received from the destination host $\mathcal{H}$
  - validate the TPM-issued credentials sent from the destination host.
  - Evaluate the security profile of the host based on the contents of the IML and the policy stated in the *IMA policy file*;
  - Decide the trustability of the host;
  - Perform other encryption operations needed to create the token containing the nonce $\mathcal{N}'$ and hash of the VM, all encrypted with the public key PK^{Bind} of $\mathcal{H}$, denoted as $\{\mathcal{N}', H_{VM}\}_{PK^{Bind}}$.
  - Return a deterministic response with the result of the attestation to $\mathcal{H}$.
- **ima_parser** verifies the contents of the IMA file received from $\mathcal{H}$ and recalculates the extension of the hashes according to the specification of TPM v1.2

6.5 Implementation analysis

The implementation described above in chapter 6 is mostly a description of the final result of the implementation. However, some knowledge has been obtained in the process of testing different approaches, tools and software configurations in order to implement the trusted launch protocol. This section contains information about the alternative tools and platforms considered, as well as motivation for some design decisions taken in the implementation phase.

6.5.1 OpenPTS integration

The possibility of using OpenPTS has been widely explored during the implementation phase of the project. OpenPTS is a proof-of-concept implementation of the Open Platform Trusted Services specification defined by the TCG [6]. OpenPTS offers a range of features, such as reference manifest (RM) and integrity report (IR) generation from the integrity measurement log (IML), verification of the result report from IR and RM, validation engine based on a finite state machine model. The verification and validation capabilities of OpenPTS initially appeared to be applicable in the compute host integrity assessment segment of the protocol. However, an additional analysis concluded that OpenPTS does not contribute to the integrity validation model required by the protocol. In particular, OpenPTS does not have a network communication component and is designed to be deployed on the TPM-enabled host itself. Thus, in order to be used in the implementation of the trusted launch protocol, the software would have to be extended to support remote host attestation. Furthermore, OpenPTS introduces an unnecessary layer of complexity particularly through the use of policy documents that determine the conformity of the hosts' PCR measurements to set of finite state machine (FSM) models provided by the software. However, OpenPTS currently lacks any support for the update of the provided FSM models. Considering the reasons stated above, as well as in an attempt to maintain the simplicity of the implementation design, it was decided to exclude OpenPTS from the TPM software stack used in the OpenStack deployment. Furthermore, it must be noted that OpenPTS lacks any kind of support in versions newer than Ubuntu 9.04 which resulted in a set of incompatible dependencies when attempting to install on Ubuntu 10.04 or 12.04.

[6]More information is available on the project wiki http://sourceforge.jp/projects/openpts/wiki/FrontPage

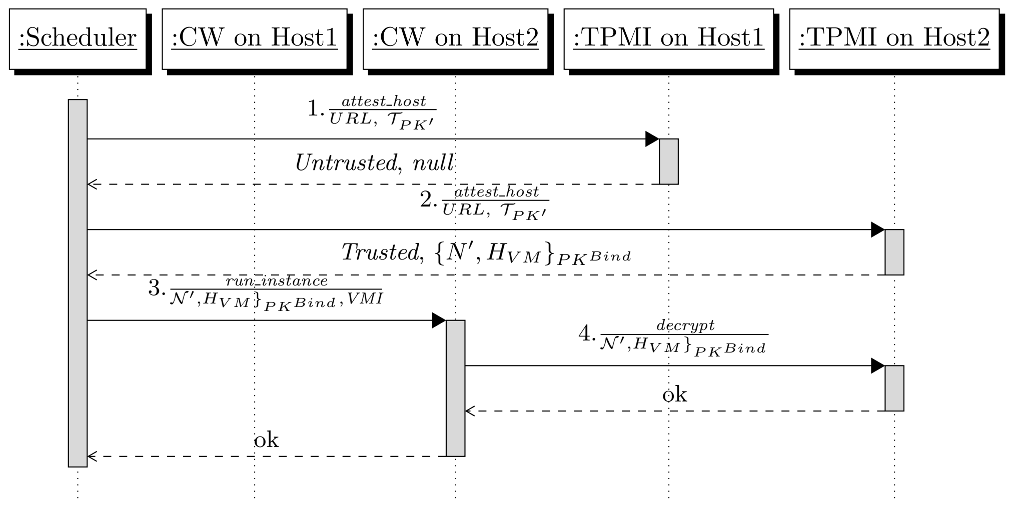

:Scheduler
:CW on Host1
:CW on Host2
:TPMI on Host1
:TPMI on Host2
1. attest_host / URL, $\mathcal{T}_{PK'}$
Untrusted, null
2. attest_host / URL, $\mathcal{T}_{PK'}$
Trusted, $\{N', H_{VM}\}_{PK^{Bind}}$
3. run_instance / $\mathcal{N}', H_{VM}\}_{PK^{Bind}}, VMI$
4. decrypt / $\mathcal{N}', H_{VM}\}_{PK^{Bind}}$
ok
ok

Figure 6.4: Host attestation call from the scheduler; 'CW' stands for "Compute Worker"

6.5.2 Platform choice and supported software

The decision to use Ubuntu long term support (LTS) versions 10.04 and 12.04 has proven beneficial for the implementation phase, due to both the relative stability of the LTS Ubuntu releases, as well as the support for OpenStack available in Ubuntu 12.04 LTS. Furthermore, Devstack support for Ubuntu 12.04 made the deployment of a multinode OpenStack environment significantly easier.

Alternatives were primarily distributions of Fedora core (Fedora 12, Fedora 16) as well as other versions of Ubuntu which have certain support from both the OpenStack and the TPM-TCG communities.

On the other hand, installation of GRUB-IMA was made significantly more difficult by the fact that support for GRUB-IMA is only available for GRUB-0.97, which has been discontinued and is no longer shipped since Ubuntu version 9.04.

Thus, while LINUX-IMA, tpm-tools and support for OpenStack are largely available in the latest LTS distributions of Ubuntu (10.04, 12.04) the lack of support for GRUB-IMA in the current version to GRUB (v1.98) is a significant barrier to the deployment of the protocol across TPM-enabled hosts.

6.5.3 Protocol implementation

Translation of the platform-agnostic trusted launch protocol described in Chapter 5 into an OpenStack-specific implementation did not require any significant changes in the protocol sequence. An exception in this case is the communication between the nova-scheduler and the `TPMI` component. Thus, according to the protocol the compute host is the OpenStack communication endpoint which communicates with TTP through the Cloud Computing Platform – TPM integration module, `TPMI` However to account for the specifics of the OpenStack architecture, the scheduler performs an initial call to attest a selected host and retries in case the host could not be validated by the TTP. Once the host has been declared trusted by the TTP the scheduler transfers the VM image and the $\mathcal{N}', \{H_{VM}\}_{PK^{Bind}}$ token obtained as a result of successful host attestation.

This protocol adaptation can be seen as an optimization rather than a strict implementation necessity. The benefit of deciding the trustworthiness of the host at the scheduling step is that in case the TTP does now acknowledge the host as trusted, or the trust level is below the one required by the client, the scheduler will be able to retry the operation with a different host, as shown in figure 6.4.

Had the protocol been implemented without modification, as shown in figure 6.5 the scheduler would

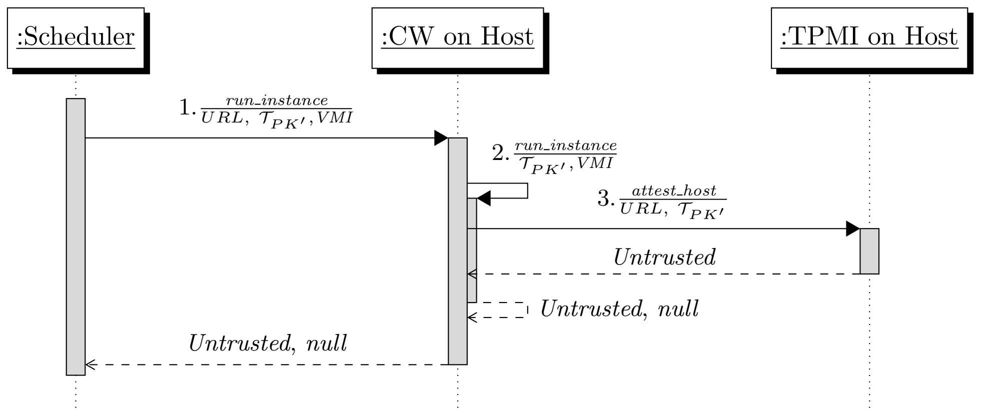

:Scheduler
:CW on Host
:TPMI on Host
1. $\frac{run\_instance}{URL,\ \mathcal{T}_{PK'}, VMI}$
2. $\frac{run\_instance}{\mathcal{T}_{PK'}, VMI}$
3. $\frac{attest\_host}{URL,\ \mathcal{T}_{PK'}}$
Untrusted
Untrusted, null
Untrusted, null

Figure 6.5: Host attestation call from the scheduler; 'CW' stands for "Compute Worker"

not be able retry the launch operation. That is because the VM launch call from the scheduler to the compute worker is asynchronous (followed by returning an `'ok'` acknowledgement to the dashboard, while the verification of whether the launch has actually succeeded or not occurs later, in a separate process. Thus, in the current implementation of OpenStack, the scheduler looses the control over the launch process at the point of calling the compute worker. Therefore, it has been decided to maintain the ability of the scheduler to retry the launch in case the host does not pass the TTP attestation process. An additional benefit is that the VM image will not have to be transferred to the host until after it has been included in the client's security perimeter.

Note that this does not compromise the security features of the trusted launch protocol, since the bind keys are generated for the specific host being attested and only a trusted host has the ability to decrypt the TTP token containing the client secret and the bind keys.

6.5.4 IMA measurement verification

As mentioned in chapter 2, the Linux-IMA kernel module, together with the GRUB-IMA patch supports the collection of runtime measurements throughout the boot sequence of the host. The measurements are collected on a granular, per-file level throughout the booting sequence. The measurements represent hashes of the contents of the loaded or executed datafile, along with a *template hash*, containing the name of the datafile. This data is both used in the calculation of the hash used to extend PCR10, as well as stored in ASCII format and binary format for subsequent verification and validation. Appendix B contains a more detailed description of the contents of the IMA measurement files.

While the number of entries in the measurement list can vary, the booting of Dell PowerEdge T310 running Ubuntu 12.04 resulted in 4304 measurement entries. The integrity of the entries were verified by the Trusted Third Party prototype, using the trusted computing principles implemented in the IMA test programs, part of the Linux Test Project. The host integrity verification consisted of two distinct steps:

1. Re-calculate the boot aggregate based on TPM's `binary_bios_measurement` provided by the host, and compare it with the `boot aggregate` entry of the `ascii_runtime_measurement`. The two values are expected to be identical.
2. Re-calculate the aggregate of the measurements stored in the `binary_runtime_measurement` file and compare it with the entry stored in PCR10. The two values are expected to be identical.

A failure to match the "expected" hash value at any stage automatically implied a failure of the attestation process. However, it must be noted that a separation of measurements into *bios* measurements and *runtime* measurements provides additional granularity that allows to determine at which stage of the boot process the configuration of the host diverged from the expected configuration.

CHAPTER 7
Conclusion

7.1 Thesis goals and process

Broadly considered, the aim of this thesis has been to examine the possibilities to increase security (in its broadest sense – confidentiality, integrity, availability) of virtualized environments in public cloud computing. Three domains – trusted computing, cloud computing and virtualization technology were included in the background study phase. While each of the three domains is actively evolving as a result of large numbers of industry and academic contributers, trusted computing had the advantage of being thoroughly specified and documented in detail. The security concerns that hamper the increased adoption of cloud computing abound, so this thesis has focused on establishing trust in the VM launch stage in a cloud computing environment. Based on that, two propositions regarding control over the integrity of the host and integrity of the launched VM image were formulated in chapter 3. In order to address the goal of building a chain of trust throughout the lifetime of a VM instance in a cloud computing environment, the launch phase of a virtual machine instance was closely reviewed in order to identify the means to address the two formulated propositions. A platform-independent trusted generic virtual machine launch protocol that would address the issues in both propositions (integrity verification of the VM host and of the launched VM image) was designed in an iterative approach. While the protocol is considered platform-independent in terms of the applicable cloud computing platform, it relies heavily on the functionality provided by the Trusted Platform Module v1.2.

In order to verify the implementability of the protocol and identify potential areas of improvement, a detailed applied implementation design has been developed based on the OpenStack cloud computing platform.

As a result, the thesis has achieved its aim by designing and implementing a trusted launch protocol for generic VM instances in public cloud computing environments, something which can be seen as a contribution towards a more secure virtualized cloud computing environment. The result fulfills all of the requirements defined in the problem statement of the thesis. However, the test infrastructure for the evaluation of the protocol performance and scalability is currently in a development phase, hence no such results are available.

In a broader sense, the thesis has addressed trust issues within virtualization environments in public cloud computing. The results of the thesis can be seen as a first contribution towards an implementation of a trusted generic VM launch protocol using an open source cloud computing platform.

In contrast with the background study and the platform-agnostic protocol design, the implementation phase of the thesis has been more iterative and exploratory. On the one hand, that is due to the immature state of the trusted computing software stack (unstable both in terms of software quality and in terms of support for the current versions of Linux-based operating systems distributions). On the other hand, the reason was partly due to the rapid development of OpenStack and introduction of new features and bugs introduced as a result.

Contribution analysis

The overall contribution on the thesis has both a theoretical and a practical aspect. An overview of the state of the art of security aspects in cloud computing and a detailed trusted launch protocol for generic VM launch in cloud computing environments can be named as the theoretical contributions.

Two practical contributions are included, namely a detailed design for the implementation of the launch protocol in OpenStack and an ongoing implementation (the results of which will be reported in an updated version of this document). The detailed implementation design can be used in order to directly implement the protocol in a deployed OpenStack environment. The results of the implementation of the protocol will be provided in an updated version of this paper, along with a performance evaluation of the protocol once the implementation phase is concluded.

Several findings of the thesis should be named. First, *trusted computing can be used to address some of the security concerns in cloud computing within the security model of an untrusted cloud service provider*. However, a set of assumptions, such as e.g. availability of physical access to the data center must be fulfilled in order to ensure a trusted VM launch in a public cloud computing environment.

Secondly, while open source cloud computing systems are in active development (something which presents both challenges and opportunities), support for trusted computing from large chip manufacturers, such as Intel and AMD, as well as support for cloud computing platforms from open source operating system vendors facilitates the application of trusted computing capabilities into cloud computing.

The results of this thesis make a case for broadening the range of use cases for trusted computing by applying it to cloud computing environments. Trusted computing, when applied correctly with certain assumptions satisfied, can offer the capabilities to securely perform data manipulations on remote hardware owned and maintained by a third party with a minimal risk for data integrity.

While the introduction of a trusted VM launch protocol can be seen as a contribution towards an opinion shift in the industry regarding trusted computing, it must be complemented by secure and trust-maintaining implementations of other frequently used cloud computing operations, such as VM migration, suspension and deletion, data storage, secure credentials management, etc.

7.2 Recommendations and future research

While the secure launch protocol proposed in this thesis only covers a single use case within provisioning of VM instances in a public cloud computing environment, it is nevertheless a first step towards bridging the gap between developments within trusted computing, virtualization technology and cloud computing platforms. The results of the current thesis (both the platform-agnostic trusted VM launch protocol and the implementation design for OpenStack) can be extended, improved and applied in the process of developing a trusted virtualized environment within a public cloud computing service.

However, future research on the topic is needed in order to address the assumptions and shortcomings of the current thesis. Such future research can be grouped into three categories.

The first category broadly includes enhancement of the proposed trusted VM launch protocol and extension of the trust chain to other aspects of cloud computing. For example, application of trusted computing to develop trusted protocols for other VM instance operations (migration, suspension, etc.), data storage and virtual network communication security. Another aspect where trusted computing could be applied is maintaining the confidentiality and integrity of user updates to VM instances. Furthermore, benchmarking and performance evaluation of the trusted launch protocol under different circumstances need to be examined further.

Topics in the second category have a more narrow scope, focusing on enhancing the proposed trusted launch protocol and addressing the shortcomings and assumptions it relies on. In particular, the proposed protocol assumes that the configuration of the host is not changed after the trusted launch of the VM instance. However, even in the case of a bona fide cloud service provider, the host of the VM can be compromised using runtime process infection. Hence, a technique to enable the client to either directly or through mediated access discover such events and protect the data used by the VM instance is a promising research area.

The third category includes topics that address the question of using the attestation results, namely the design and implementation of the evaluation policies of the trusted third party. The current assumption is that the trusted third party has access to information regarding "secure" configurations and the PCR values that hosts with such configurations should present. However, if one is to take into account the diversity of available libraries, as well as the different combinations in which they can be loaded during the boot process, then verification of the PCR values (especially the values stored in `PCR10` and the reference values in `binary_runtime_measurements` becomes a less trivial task.

The immediate future steps of the current project will focus on finalizing the prototype implementation, performance evaluation of the proposed protocol and publication of the source code on a widely available web resource.

Appendix A

Note regarding availability of implementation source code

This thesis has been carried out within a collaboration project between the Swedish Institute of Computer Science and Ericsson Research.

The source code developed during the implementation of the prototype, such as the source code of the “Trusted third party” and the OpenStack to TPM integration layer source code is freely available upon request from `n.paladi@gmail.com` and will be made available on a public code repository.

APPENDIX B

Linux IMA Measurements

The measurements taken by Integrity Management Architecture are stored in both binary and ASCII forms –
`/sys/kernel/security/ima/ascii_runtime_measurements`
and
`/sys/kernel/security/ima/binary_runtime_measurements` respectively [1].

The ASCII version of the measurements can be viewed in plaintext and consists of four columns:

- *PCR number* – the number of the PCR that is extended with the measurement value. In the case of IMA, that it PCR 10.
- *Template-hash* – the combined hash of the of the contents of the file and the "filename hint" for the loaded data, expressed as `SHA1(filedata-hash, filename-hint)`, where `filename-hint` is 256-byte long, 0-padded [2].
- *Filedata-hash* – the hash of the contents of the file containing the loaded or executed data, expressed as `SHA1(filedata-hash)`.
- *Filename-hint* – the filename of the included file, or an identifier for the loaded data (as in the case of "boot aggregate", which is the resulting hash from the PCRs 0-7.

Below follows a sample fragment of the IMA runtime measurements obtained from the host where the compute node ran in the implementation setup.

[1] More information can be found in the documentation of the Linux-IMA project, http://sourceforge.net/apps/mediawiki/linux-ima/index.php?title=Main_Page

[2] See IMA source in the Linux Kernel for further details http://lxr.free-electrons.com/source/security/integrity/ima/ima.h#L47

```
PCR#           TEMPLATE-HASH                                      FILENAME-HASH                     FILENAME-HINT
10 41a9561d512b2083136570766bcb1f5e1a0c0fb0 ima 2dfc0f17704d4c8ff2a6e00578a94b6b53082218 boot_aggregate
10 7f36b991f8ae94141753bcb2cf78936476d82f1d ima d0eee5a3d35f0a6912b5c6e51d00a360e859a668 /init
10 8bc0209c604fd4d3b54b6089eac786a4e0cb1fbf ima cc57839b8e5c4c58612daaf6fff48abd4bac1bd7 /init
10 d30b96ced261df085c800968fe34abe5fa0e3f4d ima 1712b5017baec2d24c8165dfc1b98168cdf6aa25 ld-linux-x86-64.so.2
10 76f02636ffe154df933021d2aa455dcdd1cce87b ima d2f5cbd5238aed90f808e2bdcf67cd4fe615e04f libc.so.6
10 6bc3ff76e776f198fe9b5324d06e3641a5dacaf7 ima cc57839b8e5c4c58612daaf6fff48abd4bac1bd7 /bin/busybox
10 1519afcb292dd8dc0b0a4ba1951927debb7a970f ima 65030975e1f3887efd00fbb568f00409b7c256d0 arch.conf
10 3d0d130a199ea78a53fc52f4913d28f5d0da8910 ima 0ec1deb5c2338808cf9dd31a0b16473d273fb570 initramfs.conf
10 a32bcb66bbda4ab458f1c4ab1c4724562ac5ece9 ima b17c5b1ab925b292b34971c50818f3e33d0f5e24 resume
10 6f5a8c7d7b92fd1e8c9ff64f794a8013e7ea0628 ima cbefc987113f4d16ba2c53017b298f19c12de348 functions
10 7b91f6e54fd8e38c47ddb84fe0543efe2ff5e82b ima 000a0707d8fc01bc6aa1b216593950da9ef69148 ORDER
10 f5d8d16021ae68ed4ab968a7385dca7f39abd0c9 ima e39b85ca7ba79e8086fa73a5d7f25c6464d145e4 /scripts/init-top/all_generic_ide
10 1a28d1d9b535b7a027f65fb09a9a89b7575a0b08 ima 0d53c2a458d67ffebd335051cb242c8c80497b60 /scripts/init-top/blacklist
10 b12203e9cc4fa9401d1a0e806a2046344277c130 ima 43bb08be3b6f8d8c4453177b73386dd21e5de615 /scripts/init-top/udev
10 8695dbc9c21b7a9eb89b0c3231c57c06b89c0860 ima dbbaa1ac5f2ae4efbf0862697e0407e201fdfdb7 /sbin/udevd
```

Figure B.1: Sample list of IMA runtime measurements

APPENDIX C

../compute/manager.py source code fragment

_run_instance function in compute/manager.py

```
  def _run_instance(self, context, instance_uuid,
                    requested_networks=None,
                    injected_files=[],
                    admin_password=None,
                    is_first_time=False,
                    **kwargs):
      """Launch a new instance with specified options."""
      context = context.elevated()
      try:
          instance = self.db.instance_get_by_uuid(context, instance_uuid)
          self._check_instance_not_already_created(context, instance)
          image_meta = self._check_image_size(context, instance)
          self._start_building(context, instance)
          self._notify_about_instance_usage(instance, "create.start")
          network_info = self._allocate_network(context, instance,
                                                requested_networks)
...................................................................
```

APPENDIX D

Visualization of the secure VM launch protocol by Aslam et al

D.1 List of abbreviations used in the figure

- *IM* – integrity Manager;
- *CM* – compartment Manager;
- *PM* – policy Manager;
- *CredM* – credential manager
- *UC* – user client

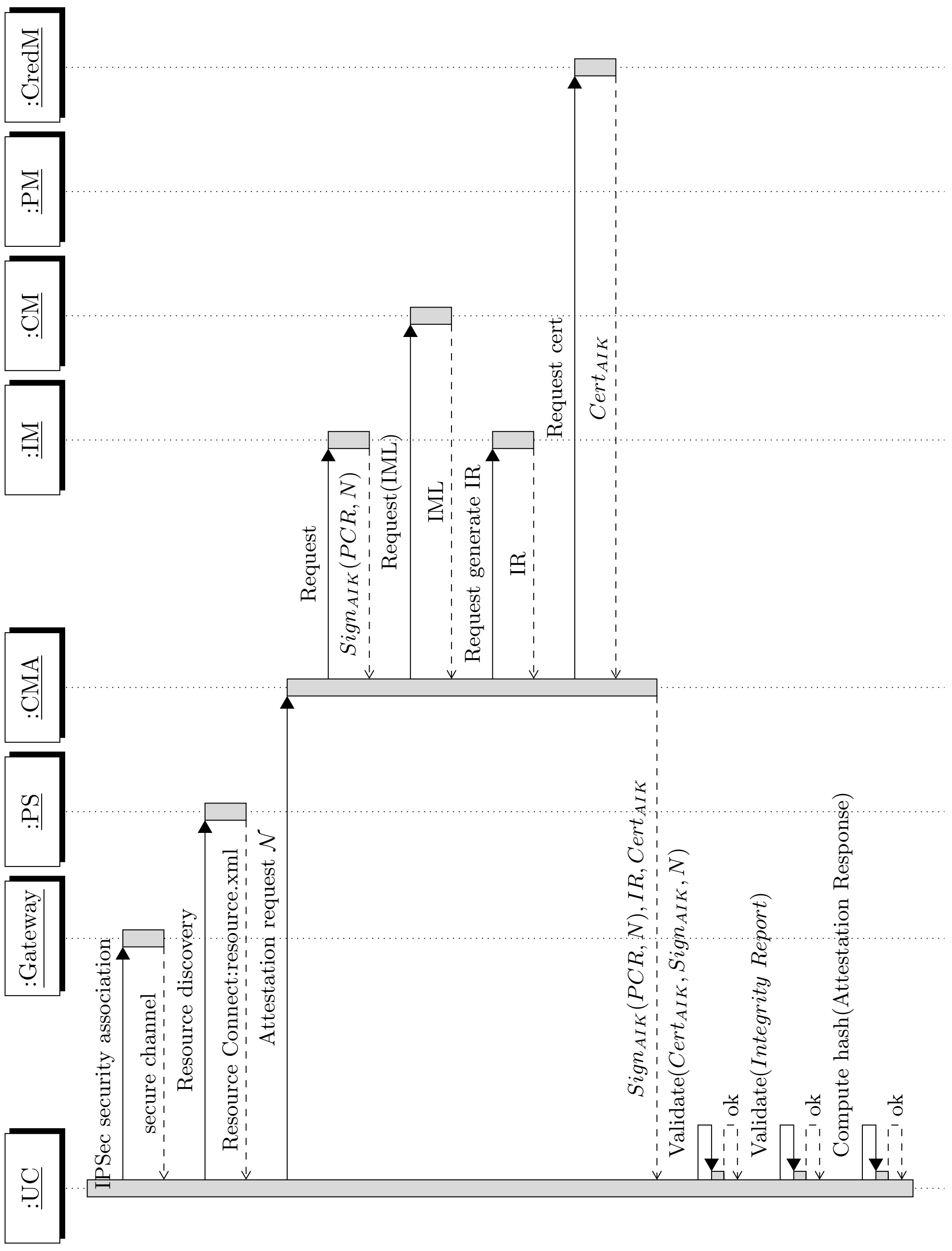
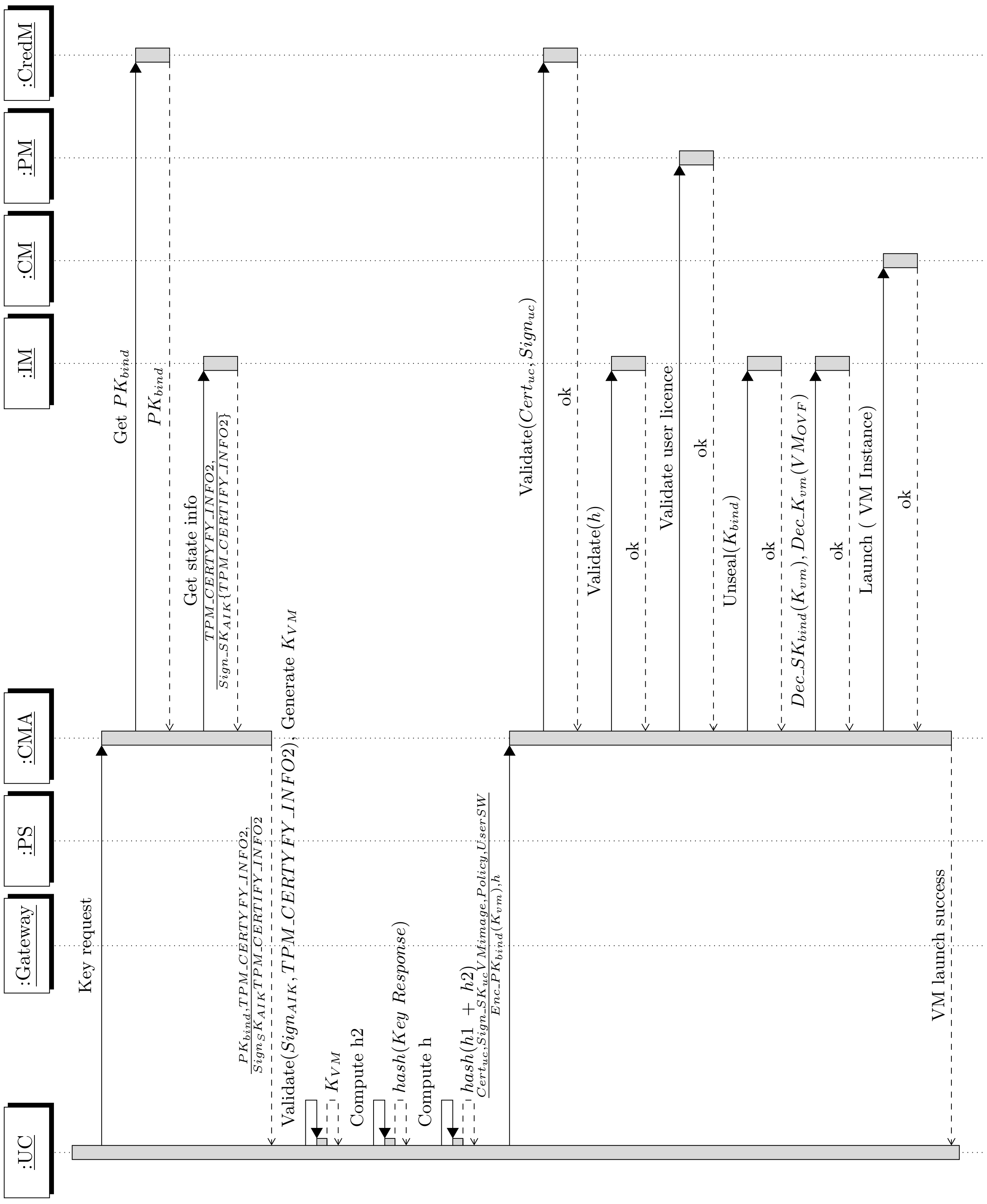

:UC
:Gateway
:PS
:CMA
:IM
:CM
:PM
:CredM
IPSec security association
secure channel
Resource discovery
Resource Connect:resource.xml
Attestation request N
Request
$Sign_{AIK}(PCR, N)$
Request(IML)
IML
Request generate IR
IR
Request cert
$Cert_{AIK}$
$Sign_{AIK}(PCR, N), IR, Cert_{AIK}$
Validate($Cert_{AIK}, Sign_{AIK}, N$)
ok
Validate(*Integrity Report*)
ok
Compute hash(Attestation Response)
ok

Figure D.1: VM Launch protocol by Aslam et al (2012), Host attestation



:UC
:Gateway
:PS
:CMA
:IM
:CM
:PM
:CredM
Key request
Get PK_{bind}
PK_{bind}
Get state info
$\frac{TPM\_CERTYFY\_INFO2,}{Sign\_SK_{AIK}\{TPM\_CERTIFY\_INFO2\}}$
Generate K_{VM}
$\frac{PK_{bind},TPM\_CERTYFY\_INFO2,}{Sign_S K_{AIK} TPM\_CERTIFY\_INFO2}$
Validate($Sign_{AIK}, TPM\_CERTYFY\_INFO2$)
K_{VM}
Compute h2
$hash(Key\ Response)$
Compute h
$hash(h1\ +\ h2)$
$\frac{Cert_{uc},Sign\_SK_{uc}VMimage,Policy,UserSW}{Enc\_PK_{bind}(K_{vm}),h}$
Validate($Cert_{uc}$,$Sign_{uc}$)
ok
Validate(h)
ok
Validate user licence
ok
Unseal(K_{bind})
ok
$Dec\_SK_{bind}(K_{vm}), Dec\_K_{vm}(VM_{OVF})$
ok
Launch (VM Instance)
ok
VM launch success

Figure D.2: VM Launch protocol by Aslam et al (2012), Secure VM Launch

Bibliography

[1] J. Somorovsky, M. Heiderich, M. Jensen, J. Schwenk, N. Gruschka, and L. Lo Iacono, "All your clouds are belong to us: security analysis of cloud management interfaces," in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, CCSW '11, (New York, NY, USA), pp. 3–14, ACM, 2011.

[2] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, (New York, NY, USA), pp. 199–212, ACM, 2009.

[3] D. Molnar and S. Schechter, "Self hosting vs . cloud hosting : Accounting for the security impact of hosting in the cloud," in *Workshop of the economics of cloud security*, pp. 1–18, 2010.

[4] Y. Chen, V. Paxson, and R. Katz, "The hybrex model for confidentiality and privacy in cloud computing," *Technical Report UCB/EECS-2010-5, EECS Department, University of California, Berkeley*, January 2010.

[5] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *Proceedings of the 2009 conference on Hot topics in cloud computing*, HotCloud'09, (Berkeley, CA, USA), USENIX Association, 2009.

[6] D. Kuhlmann, R. Landfermann, H. V. Ramasamy, M. Schunter, G. Ramunno, and D. Vernizzi, "An open trusted computing architecture – secure virtual machines enabling user-defined policy enforcement," *Work*, pp. 1–14, 2006.

[7] N. Pohlmann and H. Reimer, "Trusted computing - eine einfÃ$\frac{1}{4}$hrung," in *Trusted Computing* (N. Pohlmann and H. Reimer, eds.), pp. 3–12, Vieweg+Teubner, 2008. 10.1007/978-3-8348-9452-6_1.

[8] M. Nauman, S. Khan, X. Zhang, and J.-P. Seifert, "Beyond kernel-level integrity measurement: Enabling remote attestation for the android platform," in *Trust and Trustworthy Computing* (A. Acquisti, S. Smith, and A.-R. Sadeghi, eds.), vol. 6101 of *Lecture Notes in Computer Science*, pp. 1–15, Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-13869-01.

[9] I. Bente, G. Dreo, B. Hellmann, S. Heuser, J. Vieweg, J. von Helden, and J. Westhuis, "Towards permission-based attestation for the android platform," in *Trust and Trustworthy Computing* (J. McCune, B. Balacheff, A. Perrig, A.-R. Sadeghi, A. Sasse, and Y. Beres, eds.), vol. 6740 of *Lecture Notes in Computer Science*, pp. 108–115, Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-21599-58.

[10] R. Neisse, D. Holling, and A. Pretschner, "Implementing trust in cloud infrastructures," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pp. 524 –533, may 2011.

[11] B. Parno, “Bootstrapping trust in a ”trusted” platform,” in *Proceedings of the 3rd conference on Hot topics in security*, (Berkeley, CA, USA), pp. 9:1–9:6, USENIX Association, 2008.

[12] V. Scarlata, C. Rozas, M. Wiseman, D. Grawrock, and C. Vishik, “Tpm virtualization: Building a general framework,” in *Trusted Computing* (N. Pohlmann and H. Reimer, eds.), pp. 43–56, Vieweg+Teubner, 2008. 10.1007/978-3-8348-9452-64.

[13] A.-R. Sadeghi, C. StÃ$\frac{1}{4}$ble, and M. Winandy, “Property-based tpm virtualization,” in *Information Security* (T.-C. Wu, C.-L. Lei, V. Rijmen, and D.-T. Lee, eds.), vol. 5222 of *Lecture Notes in Computer Science*, pp. 1–16, Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-85886-71.

[14] B. Danev, R. J. Masti, G. O. Karame, and S. Capkun, “Enabling secure vm-vtpm migration in private clouds,” in *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC ’11, (New York, NY, USA), pp. 187–196, ACM, 2011.

[15] F. Rocha and M. Correia, “Lucy in the sky without diamonds: Stealing confidential data in the cloud,” in *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops*, DSNW ’11, (Washington, DC, USA), pp. 129–134, IEEE Computer Society, 2011.

[16] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A break in the clouds: towards a cloud definition,” *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 50–55, Dec. 2008.

[17] P. Mell and T. Gance, “The nist definition of cloud computing,” tech. rep., National Institute of Standards and Technology, September 2011.

[18] G. J. Popek and R. P. Goldberg, “Formal requirements for virtualizable third generation architectures,” *Commun. ACM*, vol. 17, pp. 412–421, July 1974.

[19] J. Smith and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes.* Morgan Kaufmann, June 2005.

[20] C. Wang, Q. Wang, K. Ren, and W. Lou, “Ensuring data storage security in cloud computing,” in *Quality of Service, 2009. IWQoS. 17th International Workshop on*, pp. 1 –9, july 2009.

[21] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, “Virtual infrastructure management in private and hybrid clouds,” *IEEE Internet Computing*, vol. 13, pp. 14–22, 2009.

[22] B. Rimal, E. Choi, and I. Lumb, “A taxonomy and survey of cloud computing systems,” in *INC, IMS and IDC, 2009. NCM ’09. Fifth International Joint Conference on*, pp. 44 –51, aug. 2009.

[23] H. Jin, S. Ibrahim, T. Bell, W. Gao, D. Huang, and S. Wu, “Cloud types and services,” in *Handbook of Cloud Computing* (B. Furht and A. Escalante, eds.), pp. 335–355, Springer US, 2010. 10.1007/978-1-4419-6524-0_14.

[24] W. Jansen and T. Gance, “Guidelines on security and privacy in public cloud computing,” tech. rep., National Institute of Standards and Technology, December 2011.

[25] C. Cachin, I. Keidar, and A. Shraer, “Trusting the cloud,” *SIGACT News*, vol. 40, pp. 81–86, June 2009.

[26] K. Scarfone, M. Souppaya, and P. Hoffman, “Guide to security for full virtualization technologies,” tech. rep., National Institute of Standards and Technology, December 2011.

[27] E. Gallery and C. J. Mitchell, “Trusted computing: Security and applications,” *Cryptologia*, vol. 33, no. 3, pp. 217–245, 2009.

[28] B. Parno, J. M. McCune, and A. Perrig, *Bootstrapping trust in modern computers.* Springer, 2011.

[29] T. C. Group, "Tcg specification, architecture overview, revision 1.4," tech. rep., Trusted Computing Group, 2007.

[30] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, pp. 120–126, February 1978.

[31] D. Grawrock, "Intel safer computing initiative: Building blocks for trusted computing," *Intel Safer Computing Initiative*, 2005.

[32] G. Strongin, "Trusted computing using amd "pacifica" and "presidio" secure virtual machine technology," *Inf. Secur. Tech. Rep.*, vol. 10, pp. 120–132, jan 2005.

[33] I. Press, "Openattestation," tech. rep., Intel, April 2012.

[34] B. Grobauer, T. Walloschek, and E. Stocker, "Understanding cloud computing vulnerabilities," *IEEE Security and Privacy*, vol. 9, pp. 50–57, March 2011.

[35] A. A. Raya, J. B. Alis, E. G. Herrero, and A. O. Diaz-Pabon, "Cross-site scripting: An overview.," *Innovations in SMEs and Conducting E-Business: Technologies, Trends and Solutions,*, pp. 61–75, 2011.

[36] G. Wassermann and Z. Su, "Static detection of cross-site scripting vulnerabilities," in *Proceedings of the 30th international conference on Software engineering*, ICSE '08, (New York, NY, USA), pp. 171–180, ACM, 2008.

[37] P. K. Manadhata and J. M. Wing, "A formal model for a system's attack surface," in *Moving Target Defense* (S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. Wang, eds.), vol. 54 of *Advances in Information Security*, pp. 1–28, Springer New York, 2011. 10.1007978-1-4614-0977-91.

[38] C. Li, A. Raghunathan, and N. K. Jha, "Secure virtual machine execution under an untrusted management os," in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, (Washington, DC, USA), pp. 172–179, IEEE Computer Society, 2010.

[39] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, "Eliminating the hypervisor attack surface for a more secure cloud," in *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, (New York, NY, USA), pp. 401–412, ACM, 2011.

[40] J. Kong, "Protecting the confidentiality of virtual machines against untrusted host," in *Intelligence Information Processing and Trusted Computing (IPTC), 2010 International Symposium on*, pp. 364 –368, oct. 2010.

[41] S. Berger, R. Cáceres, D. Pendarakis, R. Sailer, E. Valdez, R. Perez, W. Schildhauer, and D. Srinivasan, "Tvdc: managing security in the trusted virtual datacenter," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 40–47, Jan. 2008.

[42] J. Hendricks and L. van Doorn, "Secure bootstrap is not enough: shoring up the trusted computing base," in *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, EW 11, (New York, NY, USA), ACM, 2004.

[43] J. R. Rao and P. Rohatgi, "Empowering side-channel attacks," 2001.

[44] S. Moore, R. Anderson, P. Cunningham, R. Mullins, and G. Taylor, "Improving smart card security using self-timed circuits," in *Asynchronous Circuits and Systems, 2002. Proceedings. Eighth International Symposium on*, pp. 211 – 218, april 2002.

[45] R. Wojtczuk and J. Rutkowska, "Xen 0wning trilogy: code and demos.," 2009.

[46] R. Wojtczuk, J. Rutkowska, and A. Tereshkin, "Xen 0wning trilogy: code and demos.," 2008.

[47] B. Hay, K. Nance, and M. Bishop, "Storm clouds rising: Security challenges for iaas cloud computing," in *Proceedings of the 2011 44th Hawaii International Conference on System Sciences*, HICSS '11, (Washington, DC, USA), pp. 1–7, IEEE Computer Society, 2011.

[48] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[49] M. Aslam and C. Gehrmann, "Security considerations for virtual platform provisioning," in *European Conference on Information Warfare and Security ECIW-2011*, 2011.

[50] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *In Proc. Network and Distributed Systems Security Symposium*, pp. 191–206, 2003.

[51] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey , you , get off of my cloud : Exploring information leakage in third-party compute clouds," *Artificial Intelligence*, pp. 199–212, 2009.

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

[53] R. Holz, L. Lothar Braun, N. Kammenhuber, and G. Carle, "The ssl landscape – a thorough analysis of the x. 509 pki using active and passive measurements," in *Internet Measurement Conference 2011*, November 2-4, 2011.

[54] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Commun. ACM*, vol. 52, pp. 91–98, May 2009.

[55] J. Schiffman, T. Moyer, H. Vijayakumar, T. Jaeger, and P. McDaniel, "Seeding clouds with trust anchors," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, CCSW '10, (New York, NY, USA), pp. 43–46, ACM, 2010.

[56] M. Aslam, C. Gehrmann, L. Rasmusson, and M. Björkman, "Securely launching virtual machines on trustworthy platforms in a public cloud," in *CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science, Porto, Portugal, 18-21 April, 2012*, pp. 511–521, SciTePress, 2012.

[57] M. Price, "The paradox of security in virtual environments," *Computer*, vol. 41, pp. 22 –28, nov. 2008.

[58] C. Li, A. Raghunathan, and N. Jha, "A trusted virtual machine in an untrusted management environment," *Services Computing, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2011.

[59] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, (New York, NY, USA), pp. 85–90, ACM, 2009.

[60] G. Crnkovic, "Constructive research and info-computational knowledge generation," in *Model-Based Reasoning in Science and Technology* (L. Magnani, W. Carnielli, and C. Pizzi, eds.), vol. 314 of *Studies in Computational Intelligence*, pp. 359–380, Springer Berlin Heidelberg, 2010. 10.1007/978-3-642-15223-8_20.

[61] A.-R. Sadeghi, C. Stüble, and M. Winandy, "Property-based tpm virtualization," in *Proceedings of the 11th international conference on Information Security*, ISC '08, (Berlin, Heidelberg), pp. 1–16, Springer-Verlag, 2008.

[62] P. Goyal, "Application of a distributed security method to end-2-end services security in independent heterogeneous cloud computing environments," in *Services (SERVICES), 2011 IEEE World Congress on*, pp. 379 –384, july 2011.

[63] B. Schneier, *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. New York, NY, USA: John Wiley & Sons, Inc., 1995.

[64] M. Price, "The paradox of security in virtual environments," *Computer*, vol. 41, pp. 22–28, Nov. 2008.

[65] R. Wojtczuk, J. Rutkowska, and A. Tereshkin, "Attacking intel trusted execution technology," 2008.