D2.1
Architecture for Secure Computation Infrastructure and Self-Management of VM Security

Project number:	643964
Project acronym:	**SUPERCLOUD**
Project title:	User-centric management of security and dependability in clouds of clouds
Project Start Date:	1st February, 2015
Duration:	36 months
Programme:	H2020-ICT-2014-1
Deliverable Type:	Report
Reference Number:	ICT-643964-D2.1/ 1.0
Work Package:	WP 2
Due Date:	Oct 2015 - M09
Actual Submission Date:	2nd November, 2015
Responsible Organisation:	ORANGE
Editor:	Marc Lacoste
Dissemination Level:	PU
Revision:	1.0
Abstract:	In this document we describe the preliminary architecture of the SUPERCLOUD secure computation infrastructure and self-management architecture. We define its requirements, review the state-of-the-art, and present a first design of the proposed architecture.
Keywords:	virtualization, multi-cloud, self-management, security, hypervisor, isolation, authorization, trust management, hardware-based security, configuration compliance

This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 643964.

This work was supported (in part) by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0091.

Editor

Marc Lacoste(ORANGE)

Contributors (ordered according to beneficiary numbers)

Benjamin Walterscheid (TEC)
Alex Palesandro, Aurélien Wailly, Ruan He, Yvan Rafflé, Jean-Philippe Wary, Yanhuang Li (ORANGE)
Sören Bleikertz (IBM)
Alysson Bessani (FFCUL)
Reda Yaich, Sabir Idrees, Nora Cuppens, Frédéric Cuppens (IMT)
Ferdinand Brasser, Jialin Huang, Majid Sobhani (TUDA)
Krzysztof Oborzyński, Gitesh Vernekar (PHHC)
Meilof Veeningen (PEN)
Paulo Sousa (MAXDATA)

Disclaimer

Executive Summary

In this document we present the preliminary architecture of the SUPERCLOUD virtualization and security self-management for computation. We start by defining the design requirements of the architecture, and then review the state-of-the-art. We survey virtualization technologies and discuss designs for the virtualization infrastructure enabling the best trade-off between user control over infrastructure layers, strong security, and multi-provider interoperability. We also review isolation technologies, access control, and trust management to preserve end-to-end security between computing resources across clouds. We present a survey of security self-management, motivating the need to overcome administration complexity barriers through full security automation, seamlessly across layers and cloud provider domains. The document closes with the preliminary design of the SUPERCLOUD architecture for the virtualization and self-management infrastructure for computation, describing its different components and techniques enabling to fulfill the requirements of our design.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

The cloud is moving distributed. High maintenance costs of big data centers lead cloud architectures to evolve from centralized to hybrid, federated clouds, up to fully distributed infrastructures across heterogeneous resources, enabling optimized latencies and fine-grained geo-distribution [45, 54].
But multi-cloud interconnection is still tough to achieve [184]. *Provider-centric Distributed Cloud Computing (DCC)* [51] faces vendor lock-in, interoperability limitations, and low flexibility for customers. Therefore, DCC is now moving *user-centric* [196]. In this new paradigm, the user can instantiate and coordinate resources in a self-service manner from a number of cloud service providers.

1.1 Challenges

Beyond traditional benefits as a utility (e.g., cost reduction, scalability), the cloud comes with a promise of security and dependability. This relies on provider-based feature-rich offerings for customer VMs for protection (e.g., system intrusion monitoring, firewalling), resource management (e.g., load balancing), or availability (e.g., checkpointing/recovery). However, this promise is not fulfilled in multi-provider clouds, many such infrastructure services not being deployed uniformly.
A first problem is the *lack of user control*. Services are tightly coupled with the provider, and on its willingness to deploy them. Control is also limited by monolithic infrastructures (e.g., hypervisors hiding hardware capabilities) preventing fine-grained cloud customization by the customer.
A second problem is *interoperability*. Heterogeneity of services, and of their mapping to resources, not compatible across providers, makes achieving uniform resource SLAs difficult.
A third set of challenges deal with *security*, with three main sub-challenges:

- *Vertical challenges, i.e., security vulnerabilities in infrastructure layers*: each layer (e.g., customer VMs, provider hypervisor and services) is extremely vulnerable to attacks, in part due to new virtualization technologies. The infrastructure cannot thus be considered trusted.
- *Horizontal challenges, i.e., interoperability and unified control of security across providers.* Security component and policy heterogeneity between providers means more vulnerabilities due to mismatching APIs and workflows.
- *Complexity challenges:* administration of protection is daunting, due to the multi-provider and multi-layered nature of such an infrastructure. Unfortunately, automation of security management is still sorely lacking for the multi-cloud.

Those challenges may be summarized through 4 key objectives for multi-cloud security:

- *Self-service security*: users should control in a fine-grained manner the security of their resources.
- *Self-managed security:* the architecture should enable full automation of security management for the distributed cloud.
- *End-to-end security:* heterogeneity of security technologies should be overcome, e.g., through a distributed security abstraction layer. Trust should be managed across layers and providers.
- *Resilience:* robustness should be enhanced across clouds, to avoid reliance on a single provider.

1.2 Secure computation with self-managed protection across clouds

With the above challenges in mind for multi-cloud security – user control over security, security vulnerabilities within layers, interoperability of security mechanisms across providers, and automation of security management, both within layers and across providers – in SUPERCLOUD, we propose to develop a virtualization architecture and infrastructure enabling secure computation with self-managed protection across heterogeneous cloud providers. That infrastructure will be the refinement for computing resources of the overall SUPERCLOUD architecture and infrastructure that provides a distributed resource abstraction and flexible but unified control plane for management of security and resilience for distributed clouds.
We address self-service security and end-to-end security objectives through *a distributed virtualization infrastructure* enabling to federate multiple computing resources to build self-service clouds, the security of which is user-controlled. This computation hypervisor uses nested virtualization as core technology for implementing such security management paradigms to give sufficient control over infrastructure layers while giving strong security and multi-provider interoperability guarantees. To preserve end-to-end security between computing resources, the virtualization infrastructure also includes features for management of isolation and trust. Some of those mechanisms are based on composition of chains of trust, both horizontally across provider domains, and vertically across layers, using hardware-enabled security mechanisms.
We address self-management (and resilience) challenges through an *autonomic security monitoring infrastructure for security of computing resources*. The infrastructure enables to seamlessly detect and mitigate threats across layers and provider domains to overcome administration complexity barriers. To meet the control challenge, the self-management framework notably enables negotiation of security SLAs between user and provider, exploring relevant trade-offs. More broadly, this framework enables to orchestrate several classes of security services such as management of different types of authorizations. For safety and accountability, it also includes an automated configuration compliance mechanism for checking distributed virtualized infrastructure state correctness, or preventing violations.

1.3 Outline of the document

The rest of this document is organized as follows. In Chapter 2, we specify the design requirements of the secure multi-cloud computation and self-management infrastructure. Then we present the state-of-the-art through short survey chapters, covering virtualization technologies in Chapter 3, isolation technologies in Chapter 4, access control models, security policies, and trust management in Chapter 5, and self-management of security in Chapter 6. Finally, we present a preliminary architecture for the virtualization and self-management infrastructure in Chapter 7. We describe its different components, focusing on the techniques enabling to fulfill the requirements of our design.

Chapter 2 Design Requirements

In this chapter, we derive the design requirements for the SUPERCLOUD computing virtualization and security self-management infrastructure. We adopt a two-pronged approach to derive requirements, both from an architecture and use-case standpoints. The aim is for the infrastructure to be as flexible as possible, independently from the application domain, and while being able to support the two use-cases showcased in the project, as well as extended use cases such as NFV.
We first present a simple system model of the virtualization infrastructure (Section 2.1). We then present the design requirements for the infrastructure, from the infrastructure perspective (Section 2.2), and from the use-case perspective (Section 2.3).

2.1 System model

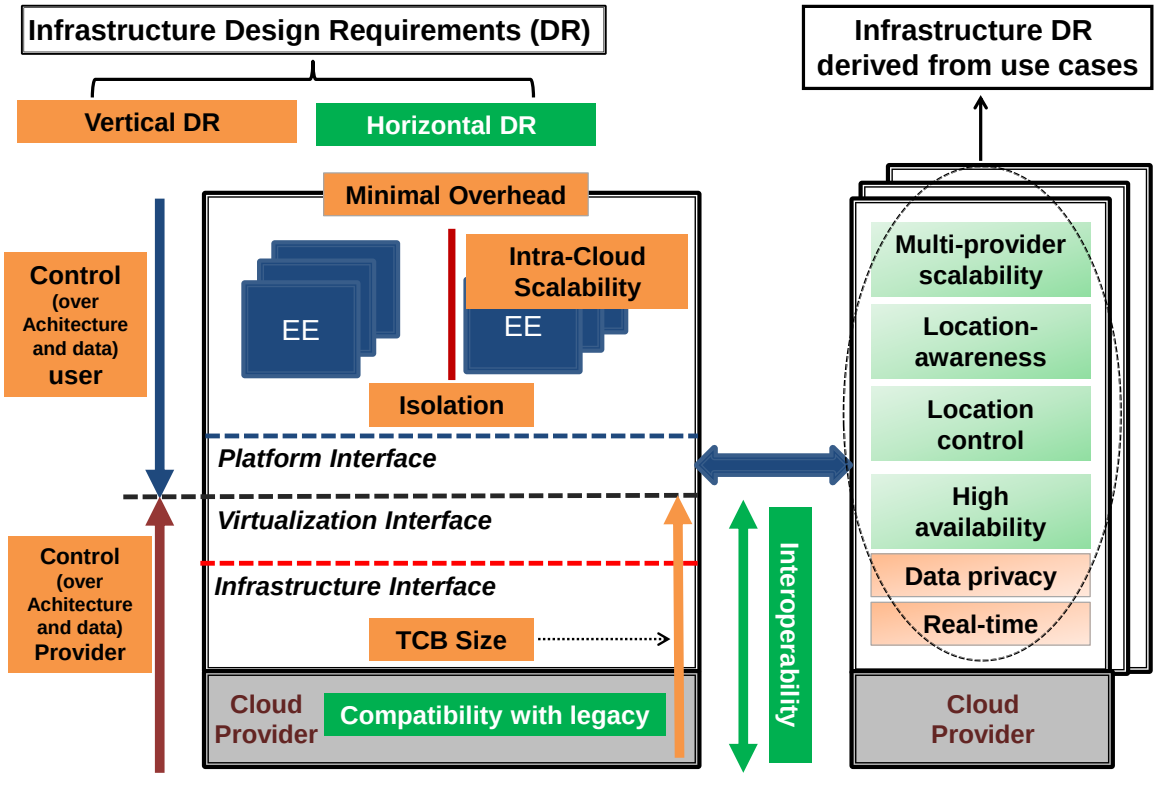


Figure 2.1: DCI system model

A simple system model for a distributed cloud infrastructure (DCI) targetted by SUPERCLOUD is shown in Figure 2.1. The DCI is composed of three layers characterized by their exported interfaces.

- The *Infrastructure Layer* enables access to hardware resources (CPU, storage, network) through an *Infrastructure Interface* (e.g., ISA) with low abstraction level and granularity.
- The *Platform Layer* provides *Execution Environments (EEs)* to the application layer through an Operating System (OS) and development framework abstraction. EEs may be for instance VMs or containers. This layer provides access to resources (e.g., processes, files) through a *Platform Interface* (e.g., syscall interface) at higher-abstraction level. This level may be highly variable, as well as interface granularity.
- The *Application Layer* contains customer software running in EEs provided by the platform.

2.2 Infrastructure design requirements

The aim is to derive design requirements on the SUPERCLOUD computing virtualization and security self-management infrastructure starting from the structure of the distributed cloud infrastructure itself. Deployment of a DCI faces major challenges that may be broadly classified as *horizontal* and *vertical*.

- *Horizontally*, DCIs lack interoperability, mainly due to heterogeneity of IaaS services, hypervisor-specific and not compatible across providers. Control is also severely limited by monolithic infrastructures preventing fine-grained cloud customization by users.
- *Vertically*, security remains the last hurdle towards wide adoption of DCIs. Due to complexity, infrastructure layers remain extremely vulnerable to attacks, making it difficult to achieve strong isolation and integrated protection.

2.2.1 Vertical design space

Vertically, a key design choice is the interface abstraction between the different layers. Such interfaces have a straightforward impact on the ability of a layer to implement a number of vertical features. From a *provider-centric* perspective, major threats [60, 109, 177] are malicious customers issuing attacks against the virtualization layer, to break isolation or perform Denial-of-Service (DoS). In this scenario, the attacker is a non-privileged malicious cloud tenant. Mitigation of such threats implies the following requirements for the virtualization infrastructure:

DR1 Isolation Tenants are not be authorized to steal or modify EE state or data from other tenants. EEs are not allowed to interfere with execution of other EEs.

DR2 Small TCB/Attack Surface The number of vulnerabilities and failures in the platform is directly linked with the code size run at the highest privilege level and the set of primitives exported. A small TCB/attack surface improves safety and integrity by design.

From a *user-centric* perspective [47, 204], major threats concern data security and privacy. Even assuming a trusted cloud provider, a malicious administrator has normally enough permissions to steal and modify customer information. Thus, provider control over the infrastructure should be limited to avoid inspection or analysis of user data and EE instances without explicit user consent. Moreover, security should be self-service, so that users can exercise fine-grained control over protection of their cloud resources according to a given security SLA. This implies the following requirement:

DR3 Control over Architecture and Data The user should be able to monitor actively its allocated resources, while enabling a high level of customizability of the architecture and its services.

Cost effective aspects such as performance and consolidation should also be considered:

DR4 Intra-Cloud Scalability Minimized EE storage and memory footprint are keys to reach good consolidation rates.

DR5 Minimal Overhead The introduced performance degradation in the architecture should be reduced to a minimum.

2.2.2 Horizontal design space

Horizontally, a key design choice is the abstraction level for interconnection in the distributed architecture. At stake is a trade-off between interconnection genericity and deployment easiness. High-level interoperability enables to realize a DCI with similar flexibility and control level as for single-provider scenarios. However, a low-level interconnection interface makes harder to conciliate different provider design choices due to lock-in and vendor specific features. Hence, the two followings design requirements:

DR6 Interoperability Usage (or migration) of resource belonging to different providers should be achieved with the same agility, flexibility, and level of control as for a single provider. This is a necessary condition for multi-provider scalability.

DR7 Compatibility with Legacy Existing user applications and management tools should be supported.

2.2.3 A summary

The overall infrastructure design requirements are summarized in Table 2.1.

Table 2.1: Infrastructure design requirements

Id	Design Requirement (DR)	Design Space
DR1	Isolation	Vertical
DR2	Small TCB Size/Attack Surface	Vertical
DR3	Control over Architecture and Data	Vertical
DR4	Intra-Cloud Scalability	Vertical
DR5	Minimal Overhead	Vertical
DR6	Interoperability	Horizontal
DR7	Compatibility with Legacy	Horizontal

2.3 Use-cases design requirements

We also derive design requirements on the SUPERCLOUD compute virtualization and self-management infrastructure starting from use-cases. We consider both: (1) use-cases coming from the healthcare domain (studied in the SUPERCLOUD project); and (2) broader application domains such as NFV. The aim is for the SUPERCLOUD architecture and infrastructure to enable secure and user-centric deployment of cloud applications, as much as possible independently from the application domain.

2.3.1 Healthcare use-cases

2.3.1.1 Medical imaging platform

There are three main Philips Healthcare use-cases that aim at deployment into SUPERCLOUD infrastructure, namely:

- Cloud data storage and disaster recovery use-case.
- Cloud data storage and processing use-case.
- Distributed cloud data storage and processing use-case.

Cloud data storage and disaster recovery use-case

The cloud data storage and disaster recovery use-case focuses on helping hospitals to ease management of the patient data stored in the hospital archive. Current hospital archives are on-premise solutions that needs to handle all the clinical data, especially imaging studies which can be as large as 1GB. Therefore, it would be attractive to offload this data into the cloud, such that storage size on-premise can be limited. For example, the data from the last 6 months is stored on premise, whilst the cloud stores for the longer period (e.g., 10+ years).
In this use-case, the cloud becomes an extension of the hospital archive. The core value of this solution is to ensure that patient data is not lost. As a result, the cloud storage is also a disaster recovery solution for the hospital. The workflow of data extends to the cloud, but performance is not the primary concern here, as patient data is always pre-fetched from the cloud to the on-premise hospital archive prior to the scheduled examination.

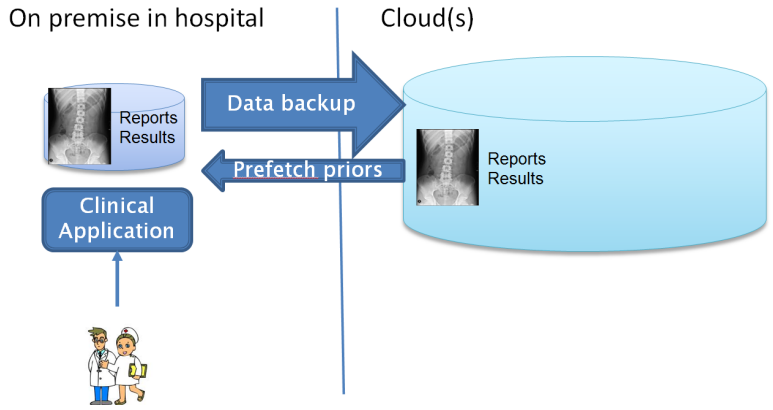


Figure 2.2: Cloud data storage and disaster recovery use-case

In general, the use-case requires:

- Privacy of medical data (top priority).
  - Storage must be robust against any security breaches (no confidentiality violation).
  - Data must not be interpretable in transit and storage, covering disk, file system and database layers.
  - Data may be de-identified, as long as data can still be fetched.
  - Role-Based Access Control (on operation level).
  - Detailed audit trails and activity monitoring.
- Correctness of stored data.
  - Data may not get tampered and must be complete and correct.
- Medical data may not cross certain legal country boundaries.
  - Guaranteed cloud storage within certain countries.
  - Distributed storage, across multiple cloud and countries, in a privacy-compliant manner; e.g., via a secure encrypted storage where data cannot be interpreted.

Cloud data storage and processing use-case

The cloud data storage and processing use-case focuses on easier access of the medical personnel to medical data processing applications. Access to applications is then possible not only from the hospital lab where the equipment is installed, but also from PCs in the hospital/home or secured mobile devices.

Thus, doctor collaboration can improve as well due to easier availability of data. The main processing of data is related to image analysis to help doctors in correct diagnosis. The image analysis is done by applying various algorithms ranging from simple image enhancing ones to detail analysis that focuses on finding potential anatomical anomalies, such as aneurysms, tumors, etc.
Since processing of the data is separated into the cloud, performance and latency are becoming critical, as imaging results and user interaction must be streamed semi real-time to the clinical user. This use-case involves only a single hospital organization.

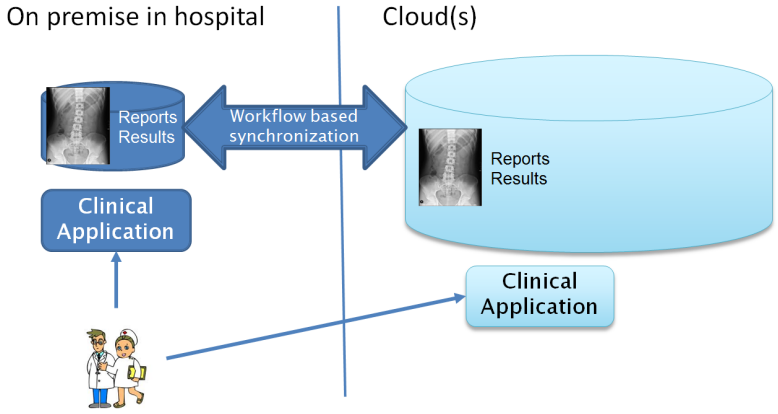


Figure 2.3: Cloud data storage and processing use-case

In general, the use-case requires:

- The same storage requirements as in the cloud data storage and disaster recovery use-case.
- Privacy.
  - Privacy during processing, i.e., applications running in the cloud may not get tampered, nor inspected for patient-related data, or processing algorithms.
  - Privacy of processed results in transit.
- Storage and processing isolation per hospital group.
  - Special permissions are required to access data outside a given hospital context (role-based access control).
- Performance.
  - Low latency between user interaction, cloud processing and updated user interface.

Distributed cloud data storage and processing use-case

The distributed cloud data storage and processing use-case focuses on easier access of the medical personnel to the medical data across hospitals, i.e., this use-case ensures patient-centric view to the healthcare professional by showing all data of the patient, i.e., not only data managed by the local hospital, but also data managed by other hospitals. This would enable providing the complete longitudinal patient record.
This use-case has highest complexity, as it involves multiple hospital organizations managing patient data. Performance and latency are critical, as imaging results and user interaction must be streamed semi real-time to the clinical user. The user may view mashup of data from multiple clouds and hospitals, or search for comparable reference studies (across the clouds), to assist in diagnosis of the treated patient. Advanced processing may even include comparing large study data, across clouds. As a result, the identity management of the clinical user is becoming critical in this use-case, as it is accessed from multiple organizations.

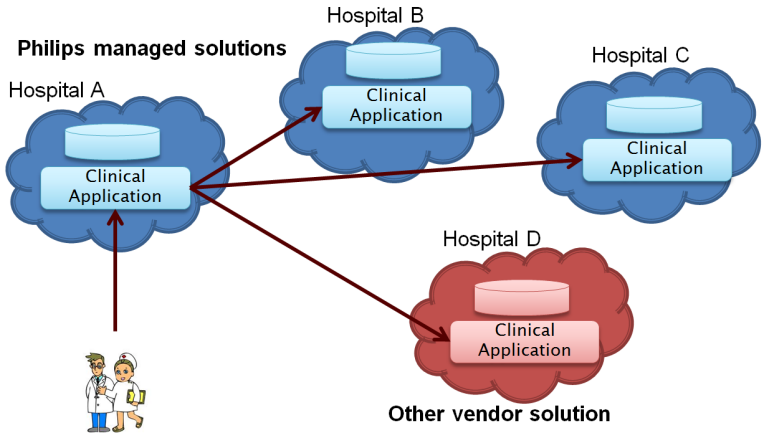


Figure 2.4: Distributed cloud data storage and processing use-case

In general, the use-case requires:

- The same storage and processing requirements as in the cloud data storage and processing use case.
- Privacy.
  - Identity management across clouds of healthcare professionals.
  - Auditing across clouds of accessed patient data.
  - Privacy of stored data, whilst access and queries are still supported.
- Interoperability across clouds.
  - Supporting Philips managed cloud solutions and 3rd party vendor solutions.
- Performance.
  - Search performance and privileges across clouds.

2.3.1.2 Healthcare laboratory information system

The healthcare laboratory information system, henceforth mentioned as CLINIdATA®LIS, is a cross-platform Web application where server components may run on any common operating system (e.g., Linux, Mac OS X, Solaris, Windows) and relational database (e.g., MySQL, PostgreSQL, Oracle, SQL Server). This system needs to integrate with dozens of other clinical and non-clinical information systems (e.g., ICU, patient identification, billing, regional health portals) and includes a set of real-time interfaces with physical electronic equipments, namely automated analysers.
CLINIdATA®LIS stores medical data along with other personal data. The SUPERCLOUD compute virtualization and security self-management infrastructure should thus comply with Directive 95/46/EC and the soon to come General Data Protection Regulation (GDPR). This implies previously stated requirements DR1 and DR2, and the following requirements:

DR8 Location-awareness Users should be aware of physical location of EEs that process user data.

DR9 Location-control Users should be able to define the set of possible physical locations, at country level, where user data may be processed.

CLINIdATA®LIS is used by different types of healthcare organizations, ranging from small laboratories with a few dozens of professionals and hundreds of transactions per day, to very large hospital clusters with thousands of professionals and tens of millions of transactions per day. Hence, these

organizations should be able to control how their resources are protected using SLAs including for instance agreed levels of availability, redundancy, load balancing, backup, disaster recovery, etc. This implies the previously stated requirement DR3.
Especially in hospitals, CLINIdATA®LIS is a critical application that needs to be constantly available in order to ensure non-stop operation of various departments including the ER (emergency room). Therefore, although concrete availability in each deployment should comply with the agreed SLA, the SUPERCLOUD infrastructure should be able to offer high availability whenever necessary:

DR10 High Availability EEs should be able to reach 99.999% availability.

As mentioned before, CLINIdATA®LIS includes a set of real-time interfaces with physical electronic equipments, namely automated analysers. These interfaces are used mostly to send commands to analysers and to receive exam results. Both communication flows have real-time requirements. This implies the following DR:

DR11 Real-Time EEs should offer real-time guarantees, namely predictable and bounded computation times.

The overall infrastructure design requirements derived from the healthcare laboratory information system are summarized in Table 2.2.

Table 2.2: Design requirements derived from the healthcare laboratory information system use case

Id	Design Requirement (DR)
DR8	Location-awareness
DR9	Location-control
DR10	High Availability
DR11	Real-Time

2.3.2 Extended use-case: NFV Infrastructure-as-a-Service

Beyond healthcare use-cases, we also consider more telco-oriented use-cases for SUPERCLOUD technology, such as application to *Network Function Virtualization (NFV)*. The NFV specification defines a three level stack [74]:

- An *NFV Infrastructure (NFVI)* provides virtual computing, storage, and network resources on top of corresponding hardware resources, multiplexed by hypervisors or network controllers.
- *Virtual Network Functions (VNF)* are telco functions (e.g., routing, firewalling...) that are virtualized, running as VM instances.
- Network services are realized as composition of several VNFs.

The NFVIaaS idea is for a third party to offer an NFVI "as a service" to service providers. Such NFVI could typically be considered as a user-centric cloud in the sense of SUPERCLOUD.
This approach expands telco reachability in locations where it maintains no physical network assets. It also significantly reduces cost and complexity of deploying new hardware or leasing fixed services.
This use-case maps the cloud service models IaaS and NaaS (Network-as-a-Service) as elements of an NFV infrastructure when provided as a service. NFVIaaS should provide computing capabilities as in a IaaS cloud and support dynamic network connectivity services similarly to NaaS. The architecture of this use-case thus combines IaaS and NaaS models to provide network services within an NFVI. Service providers can either use their own NFV/cloud computing infrastructure or leverage other service provider infrastructures to deploy their own network services, e.g. VNFs.

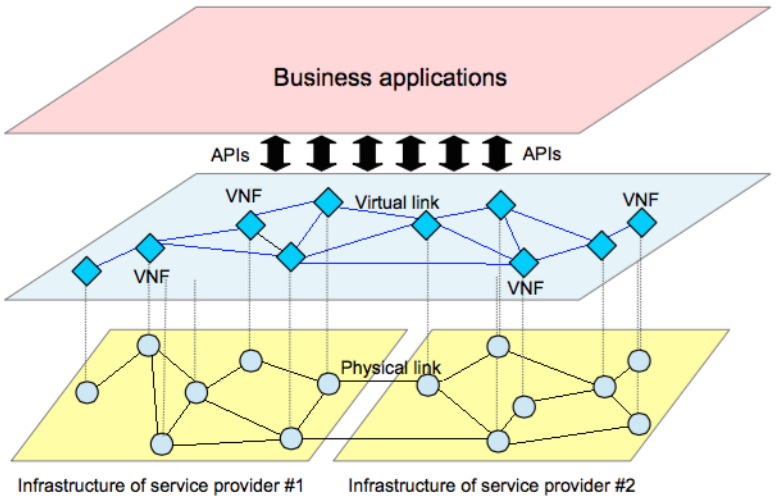


Figure 2.5: NFV use case

Figure 2.5 illustrates an NFVIaaS deployment supporting cloud applications as well as VNF instances from different service provider domains. Service Provider #2 may typically run VNF instances on the NFV/cloud infrastructure of another Service Provider #1 to improve service resilience. It may also improve customer experience by reducing latency. To perfectly comply with regulatory requirements, Service Provider #1 may require that only authorized entities can load and operate VNF instances on its NFV infrastructure. The set of resources, e.g. computing, hypervisor, network capacity, binding to network termination, that Service Provider #1 makes available to Service Provider #2 would then be constrained. Service Provider #2 is able to integrate its VNF instances running on Service Provider #1 NFV infrastructure into end-to-end network service instance, along with VNF instances running on its own NFV infrastructure.
Non-virtualized network functions can coexist with the VNFs corresponding to this use case. Virtualized network functions from multiple service providers may also coexist within the same NFV infrastructure. The NFV infrastructure also provides adequate isolation between the resources allocated to the different service providers. Thus VNF instance failures or resource demands from one service provider will not be permitted to degrade the operation of other service provider VNF instances. The NFVIaaS model thus provides basic storage and computing capabilities as standardized services over the network, storage and network equipments being pooled and made available to customers. The capability provided to customers is processing, storage, networks, and other fundamental computing resources by which customers are able to deploy and run arbitrary network services. In doing so, customers do not manage or control the underlying infrastructure, but are capable of controlling their deployed applications and can arbitrarily select networking components to achieve their tasks.

Chapter 3 A Short Survey of Virtualization Technologies

New virtualization technologies increasingly raise interest as enablers for user-centric DCC. But are such technologies enough to address both horizontal and vertical challenges and security vs. inter-operability trade-offs as needed in the SUPERCLOUD virtualization infrastructure for computation? Or are new virtualization architectures required?
This chapter attempts to provide some answers by providing a short review of the state of the art of virtualization technologies. After providing in Section 3.1, some background on virtualization principles (e.g., hypervisor architectures, virtualization types), we provide a taxonomy of the state of the art with a comprehensive review of existing designs for a distributed virtualized infrastructure, with an assessment according to Design Requirements (DRs) identified in the previous chapter. Such trade-offs will motivate the SUPERCLOUD virtualization architecture proposed in Chapter 7.

3.1 Background on virtualization

A *Virtual Machine Monitor (VMM)* is a software component able to create environments which give the impression of a real computer to all software that is executed in this environment (para-virtualization being the exception from this rule). These environments are called *Virtual Machines (VMs)*.
Usually an *Operating System (OS)* is executed inside of a VM, as on almost every computing device. An OS is also called *supervisor*, because it supervises all processes running on top of it. The VMM controls the VMs and, thereby, the OS which is running inside. In other words, the VMM supervises the supervisor, therefore it is also called *hypervisor*. The operating system instances and the applications running in a VM are called *guests*, in the context of virtualization. The physical system on which the hypervisor is executed is called the *host*.
There are two types of hypervisors, *type 1* and *type 2*, and two types of virtual machines, *para-virtualized* and *fully virtualized machines* (plus some mixed forms). These different types of hypervisors and VMs will be described subsequently starting with the hypervisor types.

3.1.1 Hypervisor types

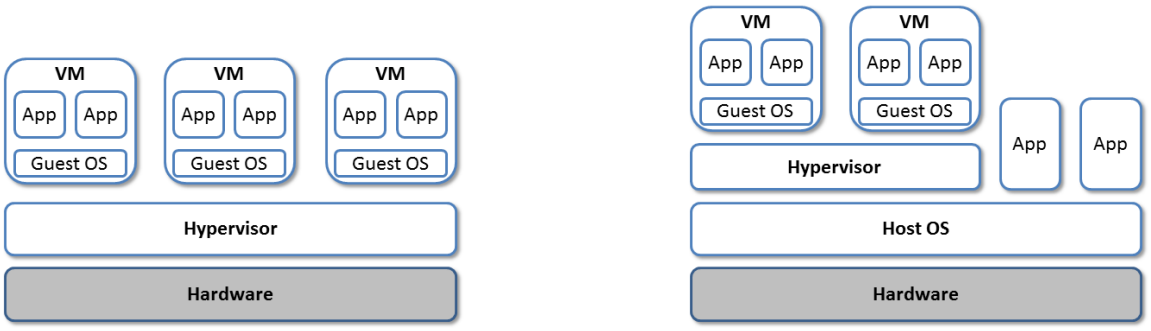


Figure 3.1: (a) type 1 hypervisor; (b) type 2 hypervisor

Traditionally, there are two types of hypervisors. One running directly on the hardware of the host (type 1) and the other running as a program on some host operating system (type 2). In a third type, mixing types 1 and 2, the hypervisor is to some degree part of the host OS. Further, some operating systems offer the means to separate groups of processes in so-called *zones* or *containers*. This is not real virtualization but has the same effect in many scenarios.

3.1.1.1 Type 1 hypervisor (bare metal)

Type 1 hypervisors (see Figure 3.1a) run directly on the hardware of the host machine. This gives the hypervisor control over all hardware. It also makes the hypervisor responsible for setting up the hardware correctly. Therefore, to be able to control the hardware it has to implement at least some of the functionality that is usually provided by the operating system. One benefit of a type 1 hypervisor is that it is faster compared the type 2 because it lacks the overhead of an extra OS running on the host machine. Another advantage is that the Trusted Computing Base (TCB) of the hypervisor is smaller compared to a type 2 hypervisor, easing the use of Trusted Computing (TC) technologies. For instance, widely adopted type-1 hypervisors are Xen or VMware ESXi.

3.1.1.2 Type 2 hypervisor (hosted)

Hypervisors of type 2 (see Figure 3.1b) are executed as a program on top of an operating system. The OS is running on the hosting machine and has full control over the hardware and is responsible for configuring it properly. The hypervisor can make use of all functionality that is offered by the operating system. This makes it more portable, since there are less hardware dependencies. The disadvantage of this approach is the overhead that is caused by the operating system running on the host machine. A popular of type-2 hypervisor is Virtualbox.

3.1.1.3 Operating system-level virtualization

Operating system-level virtualization (see Figure 3.2) cannot be clearly attributed to one of the two basic hypervisor types. As stated before, the type 2 hypervisor has the disadvantage that there is additional overhead caused by the host operating system. One approach to reduce this overhead is to move some of the hypervisor functionality into the kernel of the host operating system. This way the host OS itself is a hypervisor to some degree. If all functionality would be moved to the host OS kernel it would become a type 1 hypervisor. But some functionality remains in a user program, making operating system-level virtualization something in-between. Popular OS-level virtualization solutions are LXC, OpenVZ and Docker based on Linux Kernel or Zones for Solaris.

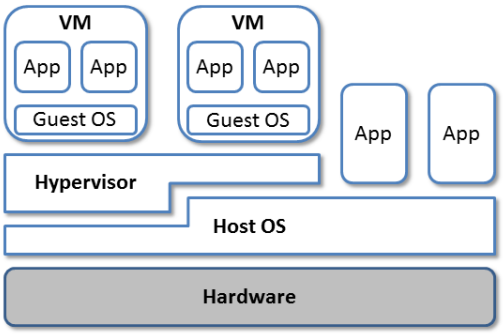


Figure 3.2: OS-level hypervisor

3.1.1.4 User-mode Linux

User-mode Linux (UML) is a version of Linux that can run on top of another Linux (the host). UML does not interact with the hardware. Instead it uses the application programming interface (API) that Linux offers to all programs. This way the UML can run as a normal process on its host (see Figure 3.3a). The UML and all other processes get isolated from each other by the host Linux like any processes.

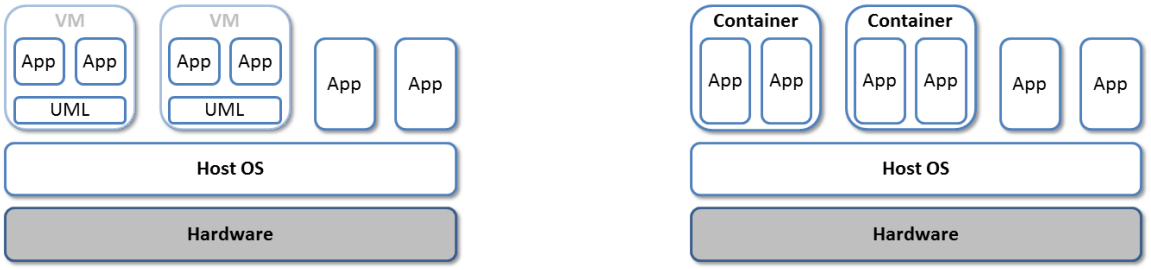


Figure 3.3: (a) User-mode Linux; (b) containers

3.1.1.5 Containers / zones

Many (UNIX-based) operating systems offer means to create some isolated areas for groups of programs (see Figure 3.3b). These zones or containers are isolated from each other by the operating system in the same way the OS isolates processes. Furthermore, these zones are separated by the use of resources, e.g., CPU time or network bandwidth can be assigned on a per-zone base. Using this mechanism, sharing resources between multiple parties can be realized without real virtualization.

3.1.2 Virtual machine types

A VM needs to provide the guest OS with all facilities it needs to run. This means it needs the ability to execute on a processor and use the memory. For the guest to provide useful services, further hardware is involved. Devices that are not strictly required to execute the basic system, (e.g., storage and network) are required. Without these devices the guest could hardly provide any valuable service. Two fundamental types of VMs that can run on top of a hypervisor: *fully-virtualized machines* and *para-virtualized machines*. An in-between type called *PV-on-HVM* is also possible.

3.1.2.1 Fully-virtualized machine

An OS running on a fully-virtualized machine does not have to be aware of the fact that it is not running on real hardware with full control over it. All hardware gets emulated in this mode and the operating system can interact with the emulated hardware as if it were real hardware. The hypervisor is transparent to the OS. For the hypervisor to be able to work like this efficiently, the hardware needs to provide some special capabilities. The need for hardware support is the reason why fully-virtualized machines are called Hardware Virtualized Machines (HVMs) in the Xen terminology.

3.1.2.2 Para-virtualized machine

An OS executed in a para-virtualized machine is aware of the hypervisor. It needs to have the ability to communicate with the hypervisor. Instead of trying to execute privileged operation it asks the hypervisor to perform them on behalf of it. This is not much different from how a user program interacts with an operating system.

3.1.2.3 PV-on-HVM drivers

The main performance bottleneck for hardware virtualized machines (HVMs) is that all device hardware needs to be emulated, causing huge overheads. To overcome this issue, para-virtualized drivers can be used. These drivers (e.g., for storage or network devices) are aware of virtualization and communicate directly with the hypervisor or another VMs.
Using PV-drivers, performance benefits of para-virtualization can be used for device hardware. At the same time there is no need to modify the operating system, when drivers can be easily installed, as in most operating systems.
Communication between PV-drivers and hypervisor works as between PV-guest and hypervisor using so-called *hypercalls*.

3.2 Virtualization architectures: a comparison

This section now attempts to compare in more detail some of those designs for a virtualization infrastructure w.r.t. the requirements seen in the previous chapter. We analyze available architecture proposals both at infrastructure and platform levels. The overall analysis is summarized in Table 3.1.

Table 3.1: Virtualization architecture comparison

Design		**Design Requirement**							References
		Vertical					Horizontal		
		Security			Performance				
		DR1	DR2	DR3	DR4	DR5	DR6	DR7	
Infra.	Plat.	**Isolation**	**TCB**	**Control**	**Scalability**	**Overhead**	**Interop.**	**Legacy**	
A. Infrastructure-based designs									
GPH	Proc.	+	+	−	−	+	−	++	[18, 167]
MH	Proc.	++	++	+	−	+	−	−−	[177]
CBH	Proc.	++	+	++	−	+	−	−	[47]
NV	Proc.	+	−	++	+	−−	++	−	[196, 24, 204]
B. Platform-based designs									
BM	Proc.	−−	−−	−−	++	+	−−	++	
BM	Lib. OS	++	++	++	++	++	−−	−−	[72]
BM	Cont.	−	−−	−−	+	++	+	++	[175]
C. Hybrid designs									
GPH	Lib. OS	+	+	−	++	+	−	−	[9, 115, 128, 132]
GPH	Cont.	++	−	−−	−	+	+	++	[7, 85]

3.2.1 Infrastructure-level designs

At infrastructure level, the following classes of solutions are available:(1) *general-purpose hypervisors*; (2) *modular hypervisors*; (3) *nested virtualization*; and (4) *bare-metal infrastructures*.

3.2.1.1 General-purpose hypervisors

The GPH is the key cloud-enabling technology. A GPH exports a hardware abstraction for concurrent execution of different OSes in isolated VM instances with an acceptable overhead (DR1, DR5, DR7). GPHs normally leverage full virtualization with hardware assistance that provides hardened resource isolation [152].
A GPH may generally be considered as monolithic and poses security concerns: GPHs have a relatively small attack surface, but non-negligible TCBs, as traditional OSes (DR2). This architecture does not

enable a customer to partially personalize services or control the hypervisor (DR3). Due to different lock-ins or implementation designs, cross-provider interoperability is often limited (DR6) [196]. The GPH traditional VM-based provisioning model suffers from slow instance deployment and scalability issues (DR4), since the considerable footprint in memory prevent massive consolidation as for containers. Recently, Google started to address such challenges with a GPH running single application-oriented VMs [167].

3.2.1.2 Minimal and modular hypervisors

The GPH monolithic design suffers from limitations in isolation and flexibility. User- and provider-centric perspectives on hypervisor architecture led to two broad classes of designs.

Micro-hypervisors (MH)

To solve the TCB size issue, the idea of MH architecture was introduced, drawing inspiration from evolution in OS architecture. Such designs were widely explored in academia [109, 177, 186], but adopted only by the mobile device industry. Within the hypervisor core modules are distinguished from non-sensitive code (e.g. device drivers). The aim is to expel as much code as possible from the TCB, making the hypervisor ultra-thin.Typically, TCB is around 10KLoC for MH, an order of magnitude smaller than a GPH [177, 186]. The whole MH architecture exposes a minor attack surface, since a significant part of services provided by the hypervisor in kernel space is now provided in user space, leaving to the MH core only the burden to correctly implement IPCs.
Some hypervisor components (e.g., device drivers, VM address space management) may be executed outside the MH, isolated, and restarted in case of compromise, improving resilience and isolation (DR2). In such increasingly modular designs, each component is provided with the privilege level required to perform its specific task, enforcing a separation of privileges. The utmost MH removes the virtualization layer altogether [109]. To preserve compatibility with legacy, TCB fragmentation principles were applied to existing GPHs [60].
However, the MH quest towards ever increasing minimalism could represent a serious limitation to preserving functions of existing platforms (DR7): the constraint of privileged code size may force MH developers to drop some basic features considered as non-essential, as multi-guest support [186].

Component-based hypervisors (CBH)

Unlike MHs, CBHs do not focus on the provider but on the user. A CBH aims to increase modularity either of the core hypervisor itself, or of the management VM [47] thanks to a component-based architecture, e.g., the Self-Service Cloud (SSC) architecture modularizes the Dom0 VM, introducing components directly controlled by the user (*user domains*) to manage resources.
The CBH offers a high level of user control (DR3), also enabling trade-offs with provider control: to let the provider monitor VM execution, inspection of user domains may be performed through mutually-trusted service domains, with known a priori and fine-grained permissions.
However, the CBH induces strong changes in control logic and APIs, impacting support of legacy cloud management platforms (DR7).

3.2.1.3 Nested virtualization

NV is a system architecture with two layers of virtualization: the guest OS virtualizes a nested guest [24]. The concept may be generalized to an arbitrary number of nested guest layers, leading to recursive virtualization [73, 79]. An NV architecture is composed of: (1) the hypervisor running above the hardware (*L0 hypervisor*); and (2) the nested hypervisor (*L1 hypervisor*). Nested VMs are generally called *L2 guests*.
The NV growing maturity opens plenty of new possibilities and avenues for research. The trend is to diversify hypervisor functionalities, with new features but also keeping existing ones. Each layer clearly addresses different sets of issues:

- L0 addresses vertical issues (DR1, DR2): it guarantees the strongest isolation, is the last line of defense of the platform, and a privileged point for monitoring its status.
- L1 deals with horizontal issues: it should provide the widest possible support for different provider platforms and virtualization techniques. This layer could be steered by the user for complete control over the infrastructure (DR3). It could also be a layer of interoperability across different providers, extending towards an integrated control plane (DR6).

Today, NV presents some limitations related to performance (DR5) and need for advanced hardware support (DR7), and TCB inherited from the use of a GPH at the L0 layer (DR2). *Performance* has long been pointed out as the main barrier to NV adoption [24]. However, hardware manufacturers have been continuously proposing new processor improvements, such as Intel VMCS shadowing to overcome such issues. Besides, for NV to be implementable in practice, the L0 hypervisor should provide an exact copy of *hardware virtualization primitives*. Full virtualization with hardware assistance is the most popular technique, but requires specific code in the hypervisor. TCB size issues are the following: each feature directly introduced within the hypervisor such as security enhancements enlarges the amount of code running in highly privileged mode and required to be trusted. In addition, some NV architectures are not yet stable and lack some of the dedicated features found in modern mainstream hypervisors [204].

3.2.1.4 Bare-metal infrastructures (BM)

A BM infrastructure where VMs run on hardware is the thinnest possible infrastructure-level design [98, 127]. It has strong benefits for isolation (DR1), TCB size (DR2), performance (DR5), but raises challenges for interoperability (DR6) and control (DR3). This design is an extreme point, raw comparison with other software infrastructure-level designs making little sense. Comparison should rather be performed with BM enhancing platform-level designs, as shown next.

3.2.2 Platform-level designs

At platform level, the following classes of solutions are available: (1) *OS processes*; (2) *Container-based architectures*; (3) *library OSes*.

3.2.2.1 OS processes

A commodity OS provides *processes* as abstraction to run user tasks and access their resources. This approach presents obvious benefits for scalability (DR4) and performance (DR5) due to lightweight execution environments.
However, suitability of processes in a multi-tenant environment is limited by the absence of resource isolation provided by the OS (DR1) – despite some Discretionary Access Control memory and file protection mechanisms. Processes are also tightly coupled with underlying OS components, and difficult to be migrated across networks (DR6).

3.2.2.2 Container-based architectures

As already seen, *containers* are user-space environments on an OS providing isolation between them and host resources [175, 67]. We focus on Linux which is the reference OS for cloud services.
After inclusion of core technologies in the Linux kernel, container architectures witnessed massive adoption by the industry to develop DevOps and "Infrastructure-as-Code" paradigms [68]. Container solutions have become prominent due to flexibility and consolidation benefits (DR4): several open-source projects such as Docker facilitate container deployment and management. Benefits also include negligible performance overheads (DR5) and high portability on homogeneous platforms (DR6).
OS-based virtualization leverages the isolation features provided by new kernel functionalities (cgroups, namespaces). But container platforms still suffer from major isolation concerns (DR1) due to Linux kernel sharing. The attack surface of a container is the Linux API, considerably larger than for

infrastructure-level designs. To reduce the attack surface, Linux kernel capability systems may be used. However, they are tricky to tune properly, and could easily lead to isolation breakouts if misconfigured. Deployment of traditional kernel security frameworks (SELinux) enforcing Mandatory Access Control (MAC) could address such concerns. Despite their effectiveness, such protection mechanisms cannot be deployed or customized flexibly from a user perspective (DR3). Moreover, an interoperability roadblock remains, due to complexity of implementing container live migration, despite early projects [61].

3.2.2.3 Library OSes

Eliminating all OS abstractions outside the kernel, *a library OS* provides a highly optimized and specialized OS architecture to execute a given application [72].
This design lifts some limitations of traditional OSes running in cloud environments such as: OS vs. GPH redundancy between system mechanisms (e.g., user management, multi-tasking); too high level OS abstractions preventing per-application performance optimization. The library OS concept was thus explored in different application domains [9, 115, 128]. For a DCA, a library OS design could therefore bring high performance and scalability benefits due to small memory and storage footprint (DR4, DR5).
An important limitation is the effort required to port applications (DR7). It may be mitigated by exporting a legacy API to applications [115]. Today, library OSes are intended mostly as minimal cloud-designed OSes to overcome several traditional limitations of commodity OSes (e.g., device driver support). Several solutions have thus a sufficient maturity level to be adopted by the industry [115, 128].

3.2.2.4 Summary

Table 3.1 summarizes our analysis, comparing three classes of existing designs according to the DRs: (A) are infrastructure-level designs, with at platform level the simplest system abstraction, standard OS processes; (B) are platform level-designs, with at infrastructure level the simplest alternative, a bare-metal configuration; (C) are some interesting hybrid designs. This table does not explore all possible architectures, but focuses on those proposed in the literature, or deployed by industrial players.
(A) The MH and the CBH are overall better than the GPH: they improve security and control, but require to rewrite user applications, forsaking legacy support. Device drivers are a major MH issue, due to costs of development, or of adaptation to a new architecture. GPH-based NV enables a smooth transition to an architecture with improved features regarding control and interoperability, but imposes heavy performance penalties. However, it is possible to achieve more secure architectures at the expense of interoperability. Today, GPH-based NV addresses either user-centric security requirements [204] or interoperability issues [195], but not both simultaneously. Doing so with would require more than two layers of virtualization [47], inducing unacceptable overheads [73].
(B) A container-based design might represent the best trade-off between legacy support, interoperability, near-optimal performance, also providing good scalability. However, security and control requirements are not completely satisfied. The library OS introduces optimized performance and high scalability, but at the cost of compatibility with legacy.
(C) A traditional GPH is deployed to overcome container isolation issues [7, 85]. However, GPH and container-based virtualization have a lot of redundant features, and user control issues are still not addressed.
In what follows, we propose some concepts for a new architecture attempting to reconcile simultaneously the above DRs. We envision a hybrid design, combining NV, MH, and CBH design principles at infrastructure level, while leaving virtualization interface flexibility to support different platform-level design alternatives such as containers or VMs, thus enabling finer-grained trade-offs depending on applications.

Chapter 4 A Short Survey of Isolation Technologies

The increasing cloud popularity has pressed the issue of privacy concerns: the cloud provider has more than necessary privileges, preventing cloud users from protecting their privacy (Section 4.1). Different classes of mechanisms are available to address this issue (Section 4.2). In this chapter, we survey three representative isolation architectures: modular hypervisor partly controlled by the user (Section 4.2.1), client-controlled secure virtual enclave based on hardware security mechanisms (Section 4.2.2), and minimization of the hypervisor TCB (Section 4.2.3). We conclude by a broader discussion of mitigation techniques for information leakage through side-channel attacks (Section 4.3).

4.1 Background

Privacy concerns are a major obstacle for many users to adopt cloud environments [87]. The cloud provider is entrusted with a large set of privileges to inspect client VM state, limiting cloud users in protecting their privacy. The cloud provider theoretically has access to client sensitive data. Any attack against or misuse of the *administrative domain* can compromise user privacy.
The administrative domain is a privileged VM that controls and monitors client VMs. To protect their reputation of running trusted cloud businesses, it can be assumed that well-known enterprises such as Microsoft and Amazon are interested in protecting privacy of their customers. While the company as a whole is interested in protecting the client data, their system administrators may have incentive to breach user privacy, e.g., pursuing monetary benefits. They may also cause a privacy breach just by mistake or accidentally.
Although many cryptographic solutions were developed to protect user confidentiality in the cloud, it is not possible to perform efficient and fast enough arbitrary computations on the data while they are encrypted, e.g., homomorphic cryptography does not offer practical performance at the current stage. Other mechanisms must thus be found to enforce protection of client data. One of the most promising approaches with regard to practicality is to *remove service provider privileges* for accessing client VM data, only keeping privileges needed for basic cloud management, i.e., VM creation/migration.
But in practice, cloud providers typically want to perform tasks such as *Virtual Machine Introspection* (VMI), running anti-virus software, or controlling client VMs for regulatory compliance. This requires the cloud provider to be able to inspect client VMs, which may conflict with client security and privacy. In commodity cloud platforms such as Xen [18] and KVM [119], the cloud provider owns the *Virtual Machine Monitor* (VMM) and an administrative domain *Dom0* (the control VM) to perform cloud management. This gives the provider full privileges over the whole platform including client VMs.
As a result, two major problems arise:

- Cloud users have no control over their own security and privacy. An insider attacker can access client data, computations, and stored cryptographic credentials, because the cloud provider is in control of client resources.

- Clients have inflexible control over their VMs. They can benefit from services enabled by virtualization, e.g., security via VM introspection [57, 83], or migration [59] and checkpointing [62]. But they depend heavily on the provider willingness to deploy these services. Such inflexibility limits clients in implementing and controlling their own security services for their VMs.

To address such issues, various approaches using *nested virtualization* (NV) [24] have been developed. NV provides the ability of running one or more hypervisors inside another hypervisor. Zhang et al. [204] proposed an approach that protects the privacy and integrity of client VMs on commodity cloud infrastructures: they separate resource management and the protection of privacy and integrity of resources owned by VMs in the virtualization layer. NV introduces a tiny security monitor underneath the commodity VMM, hence providing protection to the hosted VMs.
To address the problem of client inflexibility of control over VMs, the *XenBlanket* project [195] advocates a user-centric view of homogenization, instead of relying on providers to control client VMs. Users are enabled to run their own unmodified VMs on any cloud without any special provider support. NV is implemented by adding a virtualization layer to enable clients to avoid provider lock-in. Clients can thus implement their own services with great flexibility in terms of control over their VMs. Although those projects offer a solution to one the mentioned problems, they do not address all of them at the same time. In what follows, the most relevant works addressing these issues are described in detail, focusing on their core ideas and giving an overview on the technical approach.

4.2 Proposed solutions

Three different solutions will be presented:

- *Self-service Cloud* (SSC): administrative privileges are divided between a system-wide domain and per-client administrative domains. SSC is implemented by modifying the Xen hypervisor. It assumes that the cloud service provider is trusted, and that the physical hardware is equipped with an I/O Memory Management Unit (IOMMU) and a Trusted Platform Module (TPM) chip.
- *Cryptography-as-a-Service* (CaaS): this architecture enables clients to control provisioning and usage of their credentials and cryptographic primitives, in a protected client-specific secure execution domain. A TPM as a hardware anchor must be available on cloud nodes to verify platform integrity.
- *MyCloud architecture*: the control VM is removed from the processor root mode. Only security and performance key components are kept in the Trusted Computing Base (TCB) to minimize the attack surface.

4.2.1 Self-Service Cloud Computing

Virtual machine monitors (VMMs) are used to administer and execute client VMs flexibly. A TCB is implemented by the VMM to virtualize the underlying hardware and manage VMs. In the case of Xen and Hyper-V [187], the TCB consists of the hypervisor and an administrative domain. The administrative domain (*dom0* in the Xen terminology) is a privileged VM that controls client VMs. It has access to their VM configuration, can perform I/O for virtualized devices, and is able to monitor their physical resources. Providing these privileges to dom0 cause two major problems:

- Compromising security and privacy of client VMs by malicious system administrators, or attacks against the administrative domain because of vulnerabilities and misconfiguration.
- Inflexible control over client VMs. Clients are limited in deploying the services of their own or changing the configuration of the services offered by providers for their own purpose[1].

Butt et al. [47] introduce a new Self-Service Cloud (SSC) computing model to overcome these two problems: the power of the administrative domain is reduced, and clients have more flexibility and control over their VMs. It is done through splitting the traditional responsibilities of dom0 between the following new domains:

[1]. For example, a client might want to use a cloud security service to check for malicious network packets, but these packets are encrypted by the client, and the client is reluctant to give dom0 the permission to inspect them.

1. *Per-user administrative domain (Udom0):* This domain is used for monitoring and controlling the client VMs.

2. *Service domain (SD):* As a special-purpose user domain, it can perform privileged system services on the VMs of the client. Udom0 can delegate its privileges to SDs. SDs can then be leveraged by the client to implement services such as intrusion detection, or storage encryption.

3. *Mutually-trusted service domain (MTSD):* In practice, cloud providers must have the ability to inspect client VMs. But this compromises the privacy of the client. SSC resolves this issue by introducing MTSDs, where both the client and the provider can agree on a set of mechanisms that the provider will use to control client VMs, and that the client may verify later-on using trusted computing technologies.

4. *System Domain (Sdom0):* This is a system-wide administrative domain with privileges to start/stop Udom0 domains upon request by clients. It manages resources and runs drivers for virtualized devices. Sdom0 cannot inspect the state of the client domains, thereby ensuring the security and privacy of client VMs.

Figure 4.1 illustrates the SSC design. SSC splits the TCB of the system into a *system-level TCB*, consisting of the hardware, the SSC hypervisor, the domain builder, and the *client-level TCB*, with the Udom0 and service domain. Udom0 is the only domain which has privileges over UdomUs in its meta-domain[2]. However, to carry out specific services Udom0 can delegate specific privileges to SDs. The privileges necessary to create VMs are revoked from Sdom0, and are given to the *Domain Builder (domB)*. In SSC, the privilege model is enforced by the hypervisor to enable clients to manage their own VMs securely. This will also prevent cloud administrators to eavesdrop on client data.

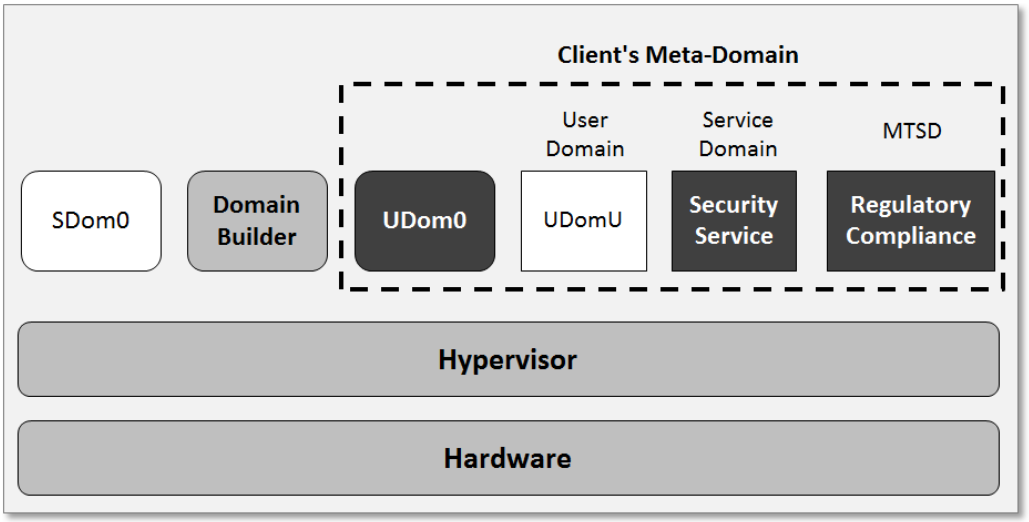


Figure 4.1: The architecture of a Self-service Cloud (SSC) computing platform [47]

To build client domains, SSC relies on trusted computing technology, such as the TPM to provide stronger security than software alone can provide. The TPM is a hardware security component that includes capabilities such as machine hardware encryption, signing, secure key storage, and attestation. By storing keys in protected hardware storage, the TPM makes encryption and signing stronger. It is assumed that the clients interact with virtual TPM (*vTPM*) [25] instances implemented in domB. The keys of this vTPM instance are bound to the hardware TPM. In order to have a hardware root of trust on each machine, SSC requires from the cloud provider that each physical machine be equipped with a hardware TPM.

[2]Meta-Domain refers to the set of client domain components.

As discussed earlier, the CloudVisor project [204] incorporates NV to protect security and privacy of clients from the administrative domain. Compared to SSC, the CloudVisor TCB is very small (5.5KLoC) formed of a small bare-metal hypervisor. The SSC TCB is very large including the entire commodity hypervisor and domB. A small TCB has a smaller attack surface and indicates more trustworthy software [82, 174]. CloudVisor uses cryptography to ensure security and privacy of client VMs. It provides protection to virtual disks owned by a VM through I/O encryption. SSC has some advantages over CloudVisor. It gives more flexibility to clients to control their VMs. It does not rely on NV which imposes overheads on client VMs. It allows cloud provider and clients to execute mutually-trusted services for regulatory compliance.

4.2.2 Client-controlled Cryptography-as-a-Service in the cloud (CaaS)

CaaS [34] is a security architecture based on Xen, where clients are in control of their credential and cryptographic primitives. Clients can establish and control Cryptography-as-a-Service in the cloud where they can securely provision keys, and even implement a private security module such as *Virtual Hardware Security Module (vHSM)* or smart card. All cryptographic operations will be executed in a protected client-specific secure execution domain. In contrast to previous works, this approach will also provide a protection for legacy VMs that are not adapted to this solution.
This solution is based on two concepts:

1. Segregating and encapsulating cryptographic operations and primitives into *a separate client-specific domain* (*domC*)[3], i.e., to isolate it from the vulnerable client VMs. domC is deployed so that it will prevent internal and external adversaries from accessing the client secrets. This protection is integrated in the entire VM life-cycle.

2. *A trusted hypervisor* that protects the separate domC against a compromised management domain effectively and efficiently. Therefore a novel security extension to the VM life cycle management is required so that it can protect domC.

In this approach, dom0 is degraded to an untrusted domain, while keeping its purpose as administrative domain. Domain management tasks are then moved to a new *trusted domain builder* (*domT*) that is privileged to build new domains. dom0 just forwards commands to domT.

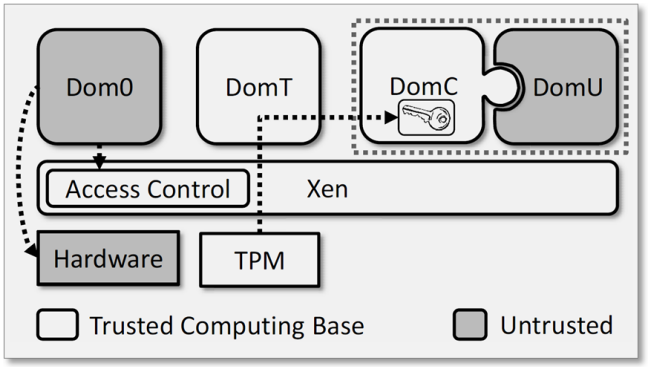


Figure 4.2: A simple illustration of CaaS [34]

In CaaS, domC is connected to domU as a Xen virtual device. It has two modes of operation: 1) *Virtual Security Module* where domC acts as security module such as an HSM and domU can use the domC interface for outsourcing cryptographic operations; and 2) *Secure Device Proxy*, which makes domC a transparent layer between domU and external devices. This layer can be used to, for example, booting a fully encrypted VM image. Both of these modes are not mutually exclusive and can be used at the same time. These two modes are illustrated in Figure 4.3.

[3]DomC is dedicated to critical cryptographic operations.

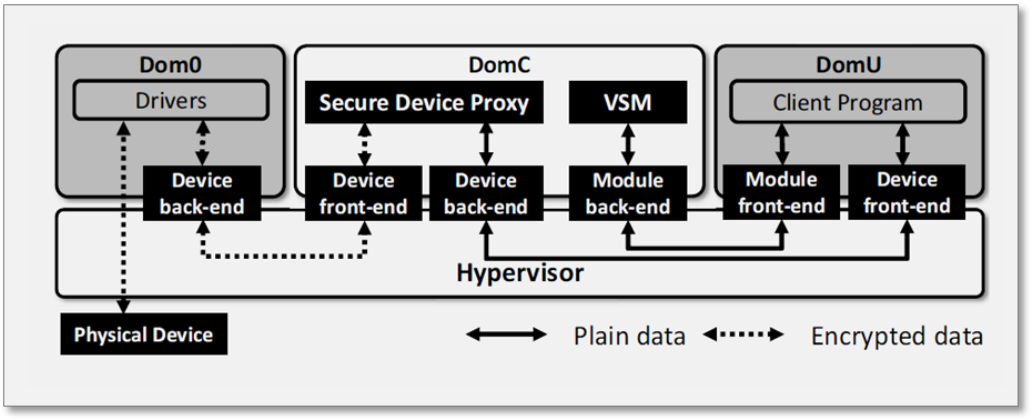


Figure 4.3: Usage modes of domC [34]

Logical isolation of domains by the trusted hypervisor and the de-privileged management domain will prevent an attacker from extracting confidential information from VMs. Empowering the domT with privileges required for domain building process, combined with the TPM based protocols ensures that Dom0 cannot access the memory of domT, domC or domU in plaintext. Additionally, since VM images are stored in encrypted and integrity protected form, any modification dom0 does to the images during launch will cause an integrity verification failure and abortion of the launch.

4.2.3 MyCloud: user-configured privacy protection in cloud computing

The TCB of SSC is too large, including the entire hypervisor and domain builder[4]. Li et al. [124] proposed *MyCloud*, a new architecture design reducing the TCB size (their prototype has 5.8K LOCs), enabling cloud provider verification of the integrity of the hypervisor. This is accomplished in a way to support customized privacy protection by the user, and simultaneously preventing the cloud provider from tampering with user privacy settings.
The control VM is removed from TCB, and has the same privileges as other guest VMs. MyCloud minimizes the privileges of the cloud provider on the hypervisor such that it cannot access client VMs memory through the control VM. Such reduction of cloud provider privileges on the hypervisor will increase security.
The primary goals of MyCloud are as follows:

1. *User-Configured Privacy Protection:* MyCloud allows users to define their own Access Control Matrices (ACM) which will be enforced by the hypervisor (TCB). ACM enables a user to define what information in his VM space can be accessed by the cloud provider. By default, all other users, including the cloud provider are prevented from accessing the user memory space. Users can decide to grant permissions to other users or the cloud provider to share information or enable specific services.

2. *TCB Minimization:* MyCloud reduces TCB size to minimize the attack surface.

[4]In dom0, usually a complete operating system runs to support user-level software. It contains many different hardware drivers from different vendors, which makes it unrealistic to formally verify the security of all of these components. The TCB can be compromised from vulnerabilities in the OS, drivers, and software components, allowing an adversary to exploit client privacy.

Components	VMM	Control VM	VM_i	ACM_i
VMM	Full	Full	Full	Full
Control VM	H	Full	A/M/D/ACM_i	R
VM_i	H		Full	R/W

Figure 4.4: MyCloud's Access Control Matrix (ACM) (A-Allocation, M-Migration, D-Deallocation, H-Hyper Calls, R-Read, W-Write) [124]

Each ACM_i of a guest VM_i is implemented by an Access Control List (ACL). ACL specifies memory regions, VM identifiers and access permissions. To request any operation, a VM initiates hypervisor calls (*hypercall*) to the VMM. VMM will check the request against the ACL of the target VM. If the access is not denied, the VMM will execute the operation on behalf of the calling VM. As depicted in Figure 4.4, VM_A modifies ACM_A through a hypercall when it wants to share any information with other VMs. Now, VM_B sends a request using another hypercall to access VM_A memory space. VMM checks this request against ACM_A to decide whether it should grant or deny access (See Figure 4.5).

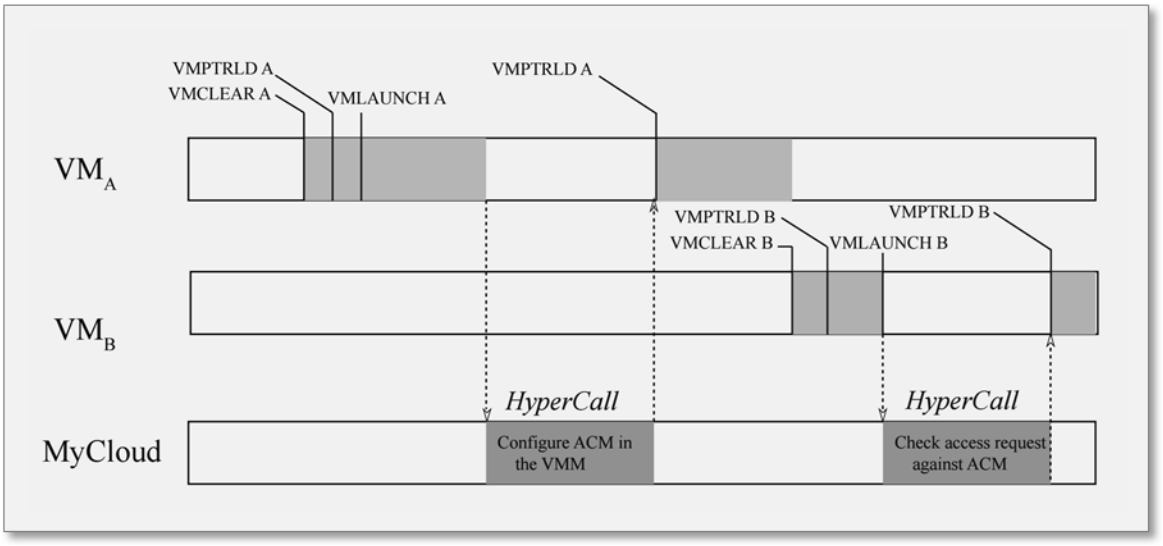


Figure 4.5: The process in which the users modify the ACM [124].

Unlike traditional architectures, the provider only controls a de-privileged control VM which is responsible for the resource management. This task is indirectly performed through the VMM interface. All of resource allocation is handled by the VMM. In the process of creating a VM, the cloud user can perform remote attestation on the platform to check its integrity and whether it is launched with known-good software components, and then negotiate a session key with MyCloud. Afterwards, the user uploads a VM image along with the hash value that is encrypted with the session key. MyCloud verifies this image, and if successful, will launch the VM and keep it running until receiving a destroy request.

4.3 Exploring information leakage in cloud platforms

The previous solutions protect against direct access through privileges. However, they do not address other methods of access such as *side-channel attacks*, or memory leakages. In cloud computing, core computing and software capabilities are outsourced on demand to shared third-party infrastructures. The way physical resources are shared between customers can create an opportunity for attackers to compromise other customers. To maximize efficiency, a cloud provider may instantiate multiple VMs

simultaneously on the same physical server. Therefore it is possible that a customer VM is executed on the same physical server as its adversary. This creates a non-obvious threat, where an adversary breaches through the isolation between VMs, and endangers the customer privacy and security.
Ristenpart et al. [157] have explored the possibility of mounting such cross-VM attacks in existing clouds. They focus on the *Amazon EC2* platform and demonstrate how it is possible to increase the likelihood of the placement of a malicious VM on the same physical server as the target VM. They also provide a simple co-residence check to detect whether two VMs are placed on a single physical machine. After a successful placement, they investigate on how to extract confidential information via cross-VM penetration.
To launch VMs to be placed on a particular physical machine, it is necessary to perform a careful empirical mapping on the cloud provider infrastructure. Mapping helps to understand where potential targets reside, and to detect the instance creation parameters necessary to attempt the placement of an adversarial instance so that it becomes co-resident with the target VM. For Amazon EC2, these creation parameters are regions, availability zones and instance types. Using network probing, it is possible to provide evidence of co-residence. The authors use the Amazon EC2 internal IP address spaces along with related instance creation parameters to create a mapping.
To detect co-residence, the authors performed a network-based co-residence check. They particularly checked the following: matching dom0 IPs addresses, small packet round-trip times, and close internal IP addresses. To make sure their check routine is correct, they have performed an experiment, where they used a hard-disk based covert channel to check whether two instances reside on one machine.
When an adversary wishes to attack an EC2 instance (an Amazon EC2 virtual machine), he should be able to arrange an instance to be placed on the same physical machine as the target. To achieve this goal, the authors have used two ways using generated mappings. First, they used the brute-force strategy where an attacker launches multiple VM probe instances over a relatively long time. Each probe instance checks whether it is co-resident with the target instance. When it is not, it will be terminated quickly. In the second strategy, the attacker takes advantage of the tendency for EC2 to launch new instances on the small set of machines, and targets recently-launched instances. It is claimed that using this approach, attacker can achieve co-residency almost half the time.
Now that it is feasible to place an attacker instance on the same physical machine as the target, it is important to investigate the methods to perform cross-VM information leakage. The authors show that (time-shared) caches give an attacker the information whether a co-resident instance is experiencing computational load or not. However, the authors only investigate those coarse-grained side-channel attacks that are not sufficient in stealing cryptographic keys. This is investigated in more detail by Zhang et al. [205], where they use cross-VM side channels to extract cryptographic keys. Zhang and Reiter [207] introduce a system called Düppel, which protects VMs from cache-based side-channel attacks in public clouds.
To mitigate such risks, cloud providers may obfuscate their internal structures as well as the placement policy to make it more difficult for adversaries to achieve co-residence with the target instances. They may also try to blind all possible side-channel vulnerabilities to minimize information leakage. The best solution might be to expose risk and placement decisions directly to users.

Chapter 5 A Short Survey of Access Control and Trust Management

The advent of cloud computing have exacerbated fundamental security challenges. For instance, answers to questions such as "how outsourced resources can be manipulated and shared while preserving their confidentiality and integrity?" become more difficult to answer when we consider the complexity of nowadays cloud infrastructures, particularly in context of multi-clouds or federation of clouds. Access control and trust management provide the necessary mechanisms to answers to such questions.

In this Chapter, we present a short literature review on existing *Access Control* and *Trust Management* models and systems. The choice to merge these two disciplines was motivated by the complementarity of the mechanisms they propose. Without and appropriate *Access Control* mechanism, a cloud infrastructure could not be *trusted* by the *customer*. Nevertheless, *Access Control* is not always enough to be trusted by *customers*. Thus, an appropriate management of both mechanisms shall be considered in SUPERCLOUD.

5.1 Access control and authorization management

Access control (AC) is the traditional mechanism by means of which software applications (originally operating systems) answer the question (i.e. request) "is the entity identified as being S can manipulate the object O via the action A?". Here the verb "can" should be understood in term of rights and not in terms of capabilities. Further, as one may notice, this question can be easily contextualised with respect to the trust issue into "Can I trust S enough to allow him performing the action A on the object O?". In this section, we present the different models that have been proposed to answer such questions. The abstract model of an access control mechanism is depicted in Figure 5.1.

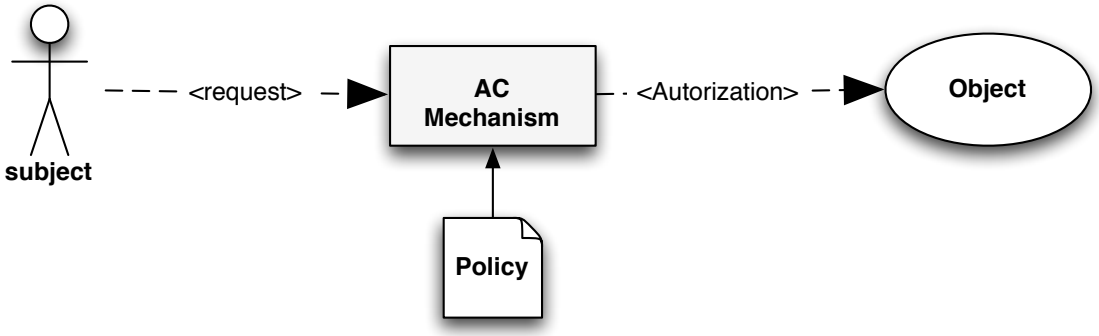


Figure 5.1: A basic access control model (adapted from [84])

- The *request* represents the type of interaction for which an authorisation is requested (e.g. read, use or login),

- The *subject*, often called *principal*, is the abstract entity (a human, a program or an artificial agent) requiring authorisation,
- The *object*[1] is the abstract artefact representing the resource the requester wants to interact with (e.g. a file, a service),
- The *mechanism* is the scheme that determines if the requester is authorised to perform the requested interaction.

Thus, the access control mechanism plays the role of guard for the manipulation of sensitive local resources. It determines if a requester should or not be authorized to manipulate the resource he requested. In the context of cloud computing, subjects are cloud users, objects are cloud resources and actions are different ways of accessing resources. A cloud user is a digital embodiment of a human, system, or service who consumes cloud resources. A cloud resource is a virtual component of limited availability within the cloud infrastructure, such as an instance, a volume, a public IP address, etc.
A cloud platform can include an authorization module to control the access to cloud resources. An authorization module is usually composed of a customizable access control policy/model and a corresponding implementation. Even in a single cloud platform, various authorization mechanisms might be applied for different cloud resources. For example, remote access to a VM is controlled by an SSH server in the VM; the network-level access is configured in various firewalls and gateways. It is an open question whether a single authorization module is needed for a cloud platform to coordinate the access to all cloud resources.
The specification of the information based on which a requester can be authorized represents permissions and is usually encoded in an *access control policy*. Based on the nature of information the policies used to specify permissions, a wide range of mechanisms have been proposed over the past decades to address the access control issue. These mechanisms can however be grouped into five categories: *identity-based*, *lattice-based*, *role-based*, *organisation-based* and *attribute-based*. The next section provides a insight on how each mechanism addresses the access control issue.

5.1.1 Identity-based access control

The general access control problem is often split into two main sub-problems: *authentication* and *authorisation*. *Authentication* aims at proving that the identity claimed by a principal is authentic, while *authorisation* tries to find whether there is a permission that allows this identity to manipulate the requested resource, and how this manipulation can be done. Identity-based Access Control (IBAC) [100, 99] has made implicit use of a closed world model, in which users and resources are *a priori* known. Under such assumptions, the problem of authorisation reduces to the one of authentication. Consequently, permissions to access a resource are directly associated with the principals identifier (e.g. user name, login, public key) as illustrated in Figure 5.2. Access to the resource (an object in Figure 5.1) is only possible when such an association exists.
Here, the information used in the policy is the *identity* of the principal requesting access to the resource. Therefore, these models are called *identity-based*. A concrete example is the implementation using access control lists (ACL). ACL are the oldest and most basic form of access control commonly found in operating systems such as UNIX. ACL are lists of principals and the actions that each principal is permitted to perform on the resource. Here the *access control policies* represent rules that determine if association between the principal and a permission exists.
In the most general form, a permission is a triple (s, o, a), stating that a user s is permitted to perform an action a on an object o. Let S be the set of all users of the system, O the set of all objects and A the set of all possible actions. The *access control policies* represent a function $f : S \times O \rightarrow A$. Consequently, $f(s, o)$ determines the list of actions that the subject s is permitted to perform over the

[1] In the remainder of this document, we will use interchangeably the terms subject and principal. We will do so for the concepts of resource and object too.

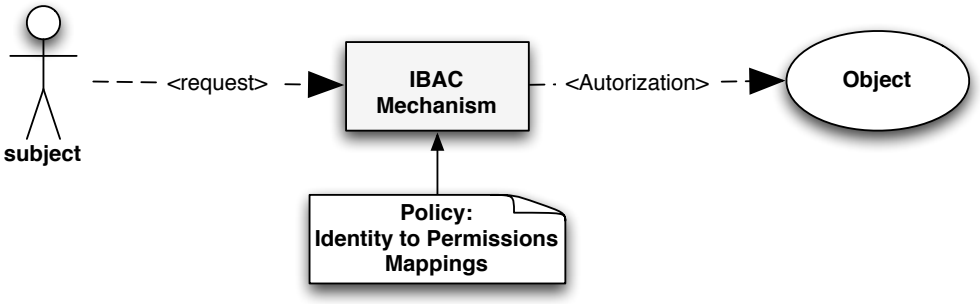


Figure 5.2: An abstract IBAC model

object o. Table 5.1 illustrates (as a matrix $A = |S| \times |O|$) the access control list of a system where $S = \{s_1, s_2, s_3\}, O = \{o_1, o_2, o_3\}$ and $A = \{a_1, a_2, a_3\}$ [70].

subject	o_1	o_2	o_3
s_1	$a_1, a_2, -$	a_2	a_2
s_2	a_2, a_2	$-$	$-$
s_3	a_1, a_1, a_2	a_1, a_2	a_1, a_2

Table 5.1: Example of an access control list

IBAC models are very easy to implement and use. However, such approaches are unable to scale when the number of users increases [203]. Moreover, the access control decisions are not related to any characteristic of the resource, the subject of the business application, making such approaches very vulnerable to attacks (ACL lists are easy to corrupt) and identity forgery (identity usurpation) [56].

5.1.2 Lattice-based access control

Unlike IBAC models, lattice-based access control (LBAC) models (also known as mandatory access control models) are deployed when the access to an object depends on its characteristics and those of the subject, and not the wills of the object owner (i.e. ACL) [163]. Subjects' and objects' characteristics are represented by security labels (or levels) that are assigned to users and resources of the system. Objects' labels reflect the extent to which a resource is sensitive while a subject's label reflects the category of objects he is permitted to access. The systems in which LBAC models are implemented are often called *multi-level security systems* as the labels used in these systems represent a partial order (e.g. Top Secret, Secret, Confidential, Unclassified) which is assumed to form a *lattice*. In LBAC, the process of access control is reduced to a control of data flow. For example, a *read* access to a resource is represented as a flow of data form the object to the subject, while a *write* access represent a flow of data from the subject to the object. In the light of this, the LBAC model's objective is to guarantee that data coming from a higher level never flows to lower level subjects, and that data coming from lowers level subject never flow up to objects of higher level. In sum, the label of a subject must be at least as high as the label of the object he wants to read, and to write on an object, the label must be at least as high as the subject's one [56]. These two security principles are respectively called "no-read-up" and "no-write-down" as illustrated in Figure 5.3 hereafter.
The Bell-LaPadula is the most famous model implementing LBAC [22]. The Bell-LaPadula model has been used in both military applications and commercial ones. Bell-LaPadula was also used to implement the security mechanisms in the Multics operating systems.
The main limit of LBAC models is their lack of flexibility and scalability. Indeed, LBAC models are quite efficient and remain relatively manageable in systems with a small number of labels.

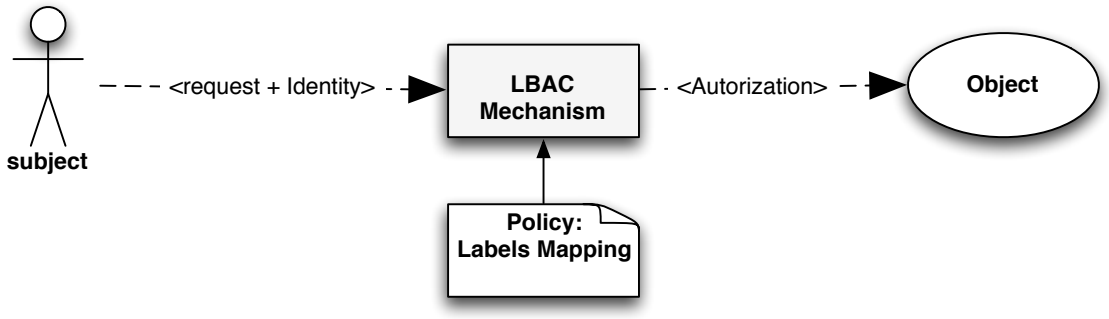


Figure 5.3: Abstract lattice-based access control model

5.1.3 Role-based access control

Both IBAC and LBAC have considerable deficiencies: LBAC models are clearly too rigid while IBAC are very hard to maintain and administrate [20]. This ascertainment has led several researchers in the early 1990s to investigate for alternative models such as *role-based access control* (RBAC) [164]. The development of RBAC was motivated by the fact that in most of the cases, sensitive resources were generally not owned by users but by the institution wherein users act in the capacity of a role of a job function [20, 200]. Therefore, the key components in RBAC are *subjects*, *roles* and *permissions* as illustrated in Figure 5.4.

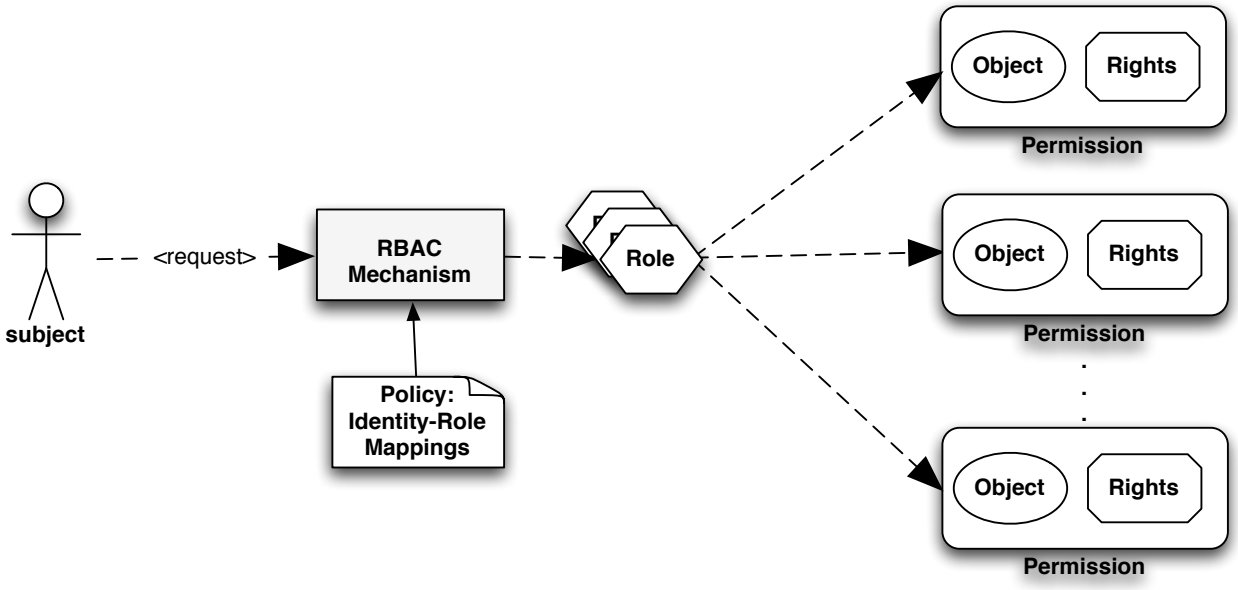


Figure 5.4: Basic role-based access control model

The policy represents here an assignment relation that associates users to the roles they hold, and roles to the permissions they are granted. Thus, roles represent an intermediary layer between subjects and permissions which make RBAC a scalable access control mechanism, and reduces considerably the complexity of access control policies specification and administration when the subjects turnover is very high. When a subject joins or leaves the system, only the link between the user identifier and the role has to be updated. Subjects are assigned roles based on their duties, qualifications or

competencies in the institution, while permissions are associated to roles based on the institution activities and goals.
RBAC received in the last twenty years considerable attention that conducted to the proposition of a whole family of models (e.g. [77, 164, 144, 162, 66, 76, 125, 41, 191, 78]). However, most of the researchers would agree on the fact that $RBAC_0$ is the core model [56, 20]. $RBAC_0$ is the main and the simplest model. $RBAC_1$ extends $RBAC_0$ with the capability to specify hierarchies of roles, introducing permissions' inheritance between roles. $RBAC_2$ extends it with constraints to enforce separation of duties, while $RBAC_3$ is a combination of $RBAC_1$ and $RBAC_2$.

5.1.4 Attribute-based access control

The main idea of *attribute-based access control* (ABAC) models is to use policies that rely on the characteristics of authorised individuals instead of their identities, roles or clearances for issuing authorisations [203, 123]. These policies are then satisfied through the disclosure of credentials issued by third party attribute certifiers (e.g. organizations, companies, institutions, etc.). Consequently, subjects can gain access to resources without being priorly known by the system administrator (or the resource owner) as illustrated in Figure 5.5.

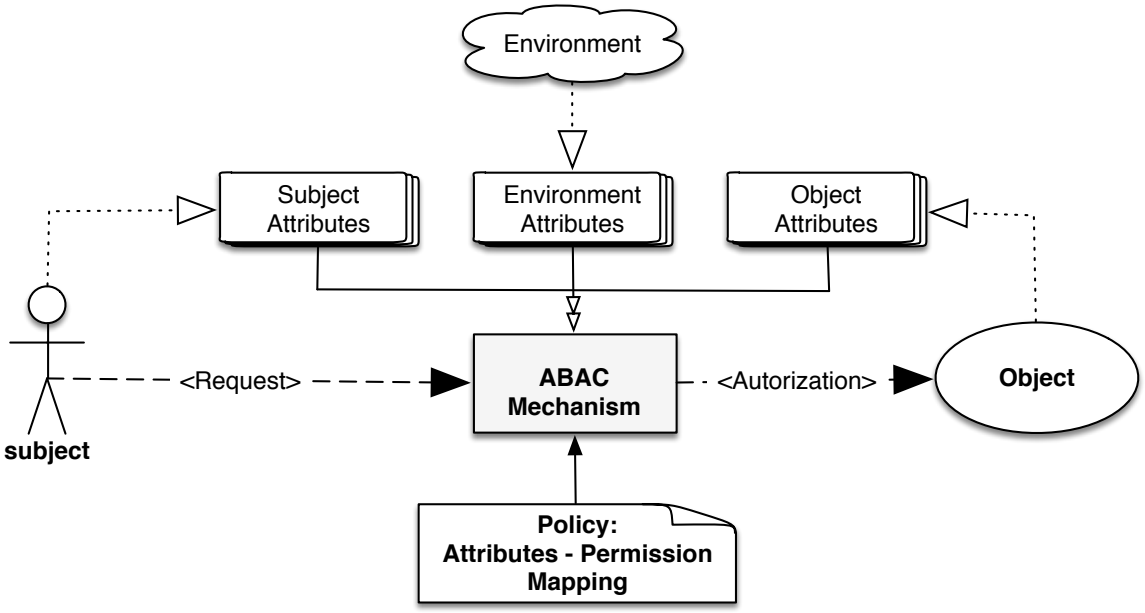


Figure 5.5: Abstract attribute-based access control model

Unlike IBAC, LBAC and RBAC, ABAC policies can define permissions based on any relevant characteristic. Characteristics fall into three categories [203]:

Subject attributes. Subjects are the entities requesting access to objects. Each subject can be characterized via a set of attributes without explicitly referring to its identity. In the ABAC literature, almost all information that can be associated to a subject is considered as an attribute. Such attributes may include subject's name, role, affiliation, address, age, and so on. Of course, the subject identity can also be considered as an attribute.

Object attributes. Objects are resources that the subject wants to manipulate. Like subjects, resources have properties that are also called attributes in the ABAC model. Resource attributes are also important in access control decision as they can affect the type of the permission accorded (e.g. a read on a text file does not have the same consequence as an execute on a program). Resource attributes may include the name of the resource, its type (text file, image, serve,

etc.), the owner, and so on. This information is generally made public and can be extracted automatically from metadata.

Context attributes. The environment or more generally the context in which the interaction is undertaken has been ignored for a long time by the security community. In previous approaches, permissions are attached to individuals (IBAC), roles (RBAC) or labels (LBAC) and derive the same authorization as long as the artefact to which they are attached remains valid (or when they are revoked by the system administrator). Thus, the context of the interaction never affects the access control decision, whereas environment attributes such as time, date, threats are relevant in applying the access control policy.

5.1.5 Organization-based access control

Organization Based Access Control (OrBAC) [107] is becoming largely used for modeling access control policies [71]. It integrates various concepts defined in the previous work such as role, hierarchy, and context.

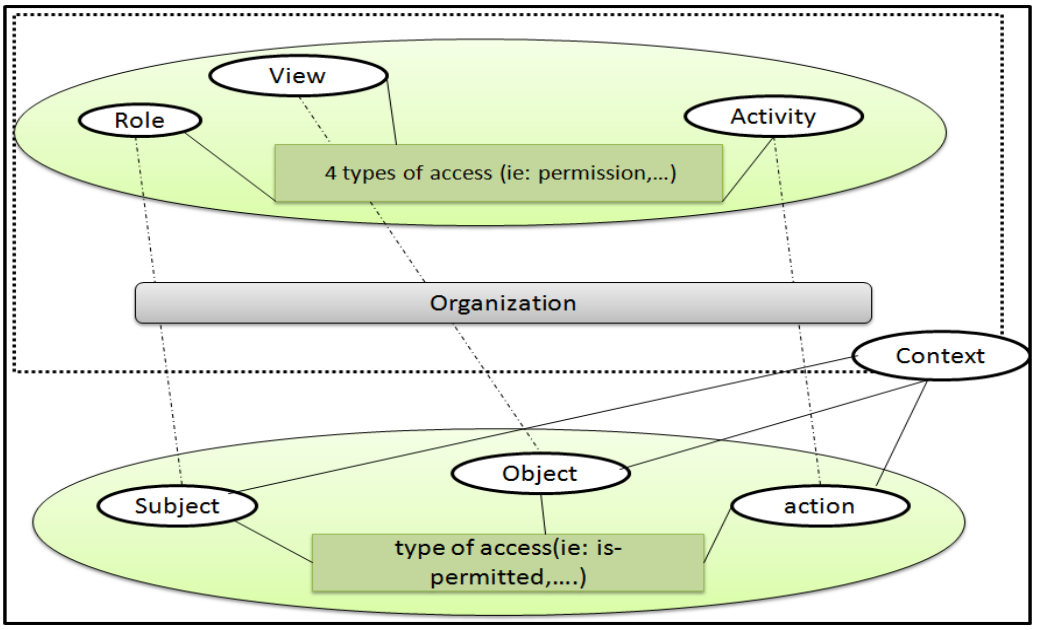


Figure 5.6: OrBAC model

The main concept of OrBAC is the entity organization. The policy specification is completely parameterized by the organization. This notion encourages researcher to handle simultaneously several security policies associated with different organizations. It is characterized by a high level of abstraction. Instead of modeling the policy by using the concrete and implementation-related concepts of subject, action and object, the OrBAC model suggests reasoning with the roles that subjects, actions or objects are assigned in the organization. Thus, a subject is abstracted into role which is a set of subjects to which the same security rule apply. Similarly, an activity and a view are respectively a set of actions and objects to which the same security rule apply.

Figure 5.6 describes the OrBAC model which introduces two security levels (concrete and abstract). OrBAC defines context concept. It is a condition that must be satisfied to activate a security rule. A mixed policy can be offered in OrBAC which define four types of access: Permission, prohibition, obligation and recommendation. Rules conflicts can appear in this policy. This problem may be resolved by affecting a coefficient to each rule. Several types of contexts can be used as temporal, geographical (physical and logical), pre-request, declared, etc. Also, we may have contexts which depend on the application. The hierarchy notion which facilitates the tasks of the administrator is also used in OrBAC.

In the same way as RBAC, two type of hierarchy (specialization / generalization and organizational) are defined. Moreover, this hierarchy can be used between different roles, different views, different activities or different contexts.

OrBAC has also its own administration model AdOrBAC. It uses the same logical formalism and the same concepts of ORBAC. As a result, OrBAC is a self-administrated model.

5.1.6 Access control In cloud computing

In cloud computing, the relationship between *subjects* and *objects/resources* is very dynamic. In addition, *subjects* and *objects* can be in distinct security domains making the specification of *access control* policies a tedious task. Consequently, many of the traditional models that we have presented previously can not be used, or requires a big adaptation effort to be in compliance with the cloud specificity. For instance, models that heavily rely on Identity (e.g. IBAC) for the management of access control reveal to be inappropriate in Cloud as they are unable to scale. Furthermore, these models can be hardly used to manage access control in *open* environments where the identity of *authorized subjects* cannot be know beforehand [200].
In this Section, we briefly review some existing approaches that tackles the challenge of managing access control in cloud environments. Most of these approaches highlight the importance of managing multi-tenant and inter-tenant access control decisions.
For instance, in [121], a flexible attribute-based multi-policy access control (ABMAC) model has been proposed. The main idea of the ABMAC is to use multiple *autonomous* security policies, a policy per tenant/domain. Then *authorization* decisions are derived by combining individual decisions. The main benefit of this model lies in its capability to scale at very large levels.
The authors of [201] proposed Access Control for Cloud Computing (AC3). This model tries to facilitate the concepts of role and task that have been used in other models. In AC3, users are classified according to their actual jobs. Thus, users will be located on a security domain that relates to their role. Every role within the model will be assigned a set of the most relevant and needed tasks for achieving this role. This approach is very similar to the principals carried out in OrBAC with a subtle analogy between tasks and activities.
Calero et al. [4] introduce an authorization system enabling cross-tenant resource access based on trust. Tang et al. [179, 181, 180] further analyzes possible types of trust relations among tenants and propose a Multi-Tenant RBAC (MTRBAC) model for collaborative access control in a multi-tenant cloud.
In [178], the authors proposed a semantic access control model for cloud infrastructures. The system has been designed for healthcare systems and aims at providing an RBAC-like model based on Semantic Web Technologies, basically ontologies.
The above selection of cloud-specific access control models should be considered in complement to several more classical approaches that only use one of the aforementioned models in cloud systems. Unfortunately, none of the above mentioned work do consider cross-tenant interaction in policy administration level.
In the next section, we motive the need for an *access control model* that goes beyond such traditional approaches. To that aim, we present the requirements that should be adhered to when designing the SUPERCLOUD *access control and authorization* framework.

5.1.7 Towards a SUPERCLOUD access control model

We conclude this chapter on access control by highlighting some desirable features for an access control model for SUPERCLOUD:

- **Expressivity** is the ability of the model to express access control policies. Expressivity is strongly related to the model. For example, RBAC use roles to group subjects. OrBAC further

enhances expressivity using activities to regroup actions and views to group objects. Developers of the cloud platform usually make adaptations to the model to better meet business requirements, which may influence expressivity. For instance, the RBAC model in the AWS platform uses roles to express delegation, which does not exist in the standard RBAC model. Expressivity may be captured through the following questions: which model has the platform adopted? what modifications were made w.r.t. the original model?

- **Granularity** is the extent to which a system can be decomposed into smaller size components. The cloud authorization system should model cloud resources with proper granularity. A coarse-grained authorization mechanism limits expressivity of the access control policy and might lead to granting unneeded access privileges. A too fine-grained mechanism might exceed the actual business requirements and increase system burden by a large amount of access control metadata.

- **Context-awareness**: context is any information that can be used to characterize the situation of an entity – a person, place, or object that is considered relevant to the interaction between a user and an application. In access control, context-awareness means that the authorization mechanism can take context information into consideration when making access decisions. Context-awareness is significant in cloud environments due to their highly distributed nature: cloud users might access the platform from different locations, during different time periods, and through different user agents. Taking those factors into consideration helps to cope with business needs and legal compliance requirements.

- **Access control model implementation** ensures usability, auditability, and scalability. The cloud authorization system should also consider availability of cloud resources.

  - **Usability** captures the user-friendly nature of the authorization system from the point of view of administrators. Policies may be counter-intuitive and unreadable. They may be also manipulated by different types of users. Some features are thus necessary to facilitate policy editing and management.
  - **Auditability**: audit represents the process to review access control records, and is useful to identify the person responsible for specific access requests or to find out violations.
  - **Scalability** means handle a growing number of access requests. This requirement is critical as a cloud platform may have a very large number of users.
  - **Resource availability**: a subject may have the privilege to access an object that is not currently available. Either access privileges and cloud resource availability may be checked separately. Or availability information may be included into the authorization module, to be taken into account in the access decision.

- **Access control administration** covers granting and revoking access privileges.

  - **Administration mode**: centralized administration where privileges are owned by one or several administrators is efficient if a cloud only has few users. However, due to multi-tenancy, a large number of users and organizations usually share a single platform. Decentralized administration where subsets of privileges may be delegated to other users might be a better choice, as each organization has its own access control requirements.
  - **Autonomous delegation** is the ability to transfer privileges between users without interference from administrators. This feature is important for the cloud, enhancing flexibility of access control while avoiding constant modification of the access control policy.
  - **Revocation** enables to cancel pre-assigned privileges once a user is no longer allowed to access a cloud resource.

5.2 Trust management

The concept of trust has been recognized as an interaction enabler in situations of risk and uncertainty. Based on this concept, several models have been proposed in the last decade. In this section, we briefly present prominent approaches of the *trust management* which encompasses *distributed artificial intelligence* and *security*.

5.2.1 Trust models in security

With the advance of Internet, the objective of researchers on security was to propose *decentralized* access control mechanisms to leverage the distributed nature of these systems. The general idea was to allow resource owners to state *who* they trust and for *which* issue. Based on this idea, several systems (called Trust Management Systems) have been proposed. We classify these models into two categories: *Decentralized Trust Management (DTM)* and *Automated Trust Negotiation (ATN)*. In the following, we present some works from both approaches.

5.2.1.1 Decentralized trust management models (DTM)

Blaze and Lacy [33] were the first to use the word *trust management* in the security domain [32]. The authors justified the use of trust in reference to the delegation mechanisms they used to address distribution in modern systems (e.g. Internet). Credentials (digitally signed documents) are used to allow an individual to express the trust relationship it has with another individual with respect to a particular issue (e.g. accessing a resource). For instance an individual A may issue a credential stating that he trusts another individual B to access a resource R he owns. Subsequently, B can express the trust he puts in another individual C with respect to the same issue. Now if C requests to access R, he has to provide the credential provided by B and the objective of the model is to evaluate whether there is a valid trust (delegation) chain from A to C about accessing the resource R.

Grandison and Sloman proposed the SULTAN (Simple Universal Logic-Oriented Trust Analysis Notation) trust model [86]. This model was developed with the objective to support secure interactions in Internet applications. Typically, decisions derived using SULTAN are about the evaluation of trust (and distrust) relationship, which has been used as a basis for developing distributed access control schemes (e.g. allowing users to access sensitive resources). SULTAN makes use of recommendation and past experience which are filtered based on rules (i.e. policies) stated by the user. The trust decision is then made based on this evaluation along with a risk approximation.

Kagal et al.[106] proposed a trust model for distributed security in the context of multi-agent systems. They proposed a flexible representation of trust relationships that they express in the form of *permissions* and *delegations*. Later, they extended their model to integrate three new forms of trust relationships called *obligations*, *entitlements* and *prohibitions*. Concretely, the notion of trust used by the authors is purely *interpersonal* as it express the degree of privilege that one individual is willing to accord to its interlocutor. Kagal and colleagues described a scheme for representing delegation and restricting re-delegation among communities.

Yaich et al.[199] developed an Adaptive trust model in which access control requests are evaluated with respect to the degree of trust associated to the requested. The novelty of this work lies in the use of meta-policies to adapt the trust evaluation scheme to context changes. The Adaptive and Socially-Compliant Trust Management System (ASC-TMS) also makes use of the Social dimension of trust decisions to improve their efficiency.

5.2.1.2 Automated trust negotiation models (ATN)

Credentials which are implemented in DTM as identity or delegation certificates are used in trust negotiation systems to convey the identity and the attributes of the holder. So releasing a credentials implies the disclosure of such sensitive information. To that aim, Winslett and colleagues [202] introduced the concept of *trust negotiation*. Trust is established through the gradual, iterative, and mutual disclosure of credentials and access control policies. This model has been proposed to leverage *privacy* issues that may arise when the disclosed credentials are sensitive. The authors build on exiting *decentralized access control models* to allow bilateral establishment of trust.

Trust-$\mathcal{X}$ is a trust management system that was designed for trust negotiation in peer-to-peer systems [27, 28]. The Trust-$\mathcal{X}$ engine provides a mechanism for negotiation management. The main strategy used in Trust-$\mathcal{X}$ consists in releasing policies to minimise the disclosure of credentials. So only credentials that are necessary for the success of a negotiation are effectively disclosed [176].

Bonatti et al.[65] proposed a flexible and expressive negotiation model called *PROTUNE* (*PRovisional TrUst NEgotiation*). This framework is a system that provides distributed trust management and negotiation [42, 44] features to web services. *PROTUNE* consists in a policy language (based on PSPL [43]) and an inference engine based on PeerTrust [141]). The most innovative aspect of PROTUNE relies in its high degree of expressiveness. One of the main advances made by PROTUNE lies in the use of *declaration* along with credentials during the policy evaluation process. *Declarations* are the unsigned equivalent of credentials. They can also be considered as statements that are not signed by a certification authority. However, the most novel part of the project remains the policy specification language which combines access control and provisional-style business rules.

5.2.2 Trust models in distributed artificial intelligence

Trust models in Distributed Artificial Intelligence (DAI) fall into three categories; *probabilistic models*, *reputation models* and *socio-cognitive* models.

5.2.2.1 Probabilistic models

Marsh's work [131] on trust represents the first comprehensive and formal model of trust using a probabilistic approach. Marsh experimented in his doctoral thesis the use of trust as a basis for cooperation among autonomous agents. Starting from the assumption that "neither full trust or distrust are actually possible" in such systems, he proposed a complex calculation algorithm that computes a continuous value of trust based on several factors. This value represents the probability that A will behave "as if" he trusts B.
Manchala developed the first trust model that explicitly uses the notion of risk [130]. Manchala specified a decision matrix in which he used the cost of the transaction, the expected outcome, and the history of the transactions. These factors are then used together in a probabilistic function to determine whether a transaction with a partner should be conducted or not. Manchalas risk-trust matrices are intuitive and simple to apply. The higher the value at stake, the more positive experiences are required to decide to trust [104].
Burnett et al. [46] used recently the concept of stereotypical trust. In their approach, an individual uses machine learning methods to build stereotypes based on the agent past experience and the partners attributes. A stereotype represents a form of reputation that associates to a class of agents (characterized by their attributes) to the behavior they tend to exhibit. These stereotypes are then used alongside with probabilistic model to make trust decisions.
Previous to this approach, in 2009, Liu et al. [126] already used stereotypes in trust decision making. However, in their approach, they used the stereotypes to predict whether the undergoing interaction of an individual will be successful or not. More recently, Fang and colleagues proposed in [75] a new

model in which they used fuzzy semantic decision tree (FSDT) learning methods to improve the ability of learning trust stereotypes based on only limited data about the partner.

5.2.2.2 Reputation models

Reputation is the social evaluation of a group, a community or a society of agents towards the trustworthiness of an individual [159]. In DAI, and more particularly in multi-agent systems, reputation has been considered as a substantial dimension of trust [105]. In the following, we review some predominant reputation models.
ReGreT [159] is a well-known decentralized trust and reputation model for e-commerce. Proposed by Sabater and Sierra in 2001, the main objective of *ReGreT* was to make more accurate trust evaluations. To that aim, the authors used three factors based on which trust was computed: *the direct experience*, *the global reputation* and an ontological *fine-grained reputation* which defines reputation values for each trait of the individual using the ontology. In ReGreT, the network to which the agent belongs is used to assess the credibility of the information provided by each agent. Social relationships are presented in the form of fuzzy rules which are later used to determine whether the witness information provided by an agent should be considered or not.

Jøsang [105] proposed a reputation model (called the Beta Reputation System) for the decision making in the context of e-commerce transactions. The authors used the concept of reliability along with the probability of success to determine the trustworthiness of a partner. The reliability of an individual is assessed in a direct and indirect way. The direct reliability is computed based on previous knowledge about the partner, while the indirect one is given by recommendation from other trust third party. The indirect reliability is then computed by making the average of all recommendation weighted by the recommender trust degree. Then this value is combined with the direct reliability in order to derive a trust degree. Once this trust degree obtained, it forms a belief that is described as set of fuzzy propositions such as "A believes that B is very trustworthy".

FIRE [97] is another important model which has been designed by Huynh and colleagues for multi-agent systems. The authors compute trust based on past experiences, the role of the agent, its reputation and a kind of certified reputation. Roles are used to determine to which degree an agent that have a certain position in the society could be trusted. The main idea is that trust depends on the fulfilment of the role ascribed to the agent. Also, the authors make a distinction between witness reputation and certified reputation. Certified reputation is a reputation that comes from certified presumably trusted witness, while normal reputation comes from every agent of the society.
Vercouter and Muller proposed the LIAR model to process recommendations among multi-agent systems [188]. The authors consider trust as a multi-factor concept; they distinguished between the trust in an agent as a partner and the trust in an agent as a recommender. This latter form of trust is used to evaluate the witness information. The aggregation of witness information is performed using a weighted average, in which the trustworthiness degree of the recommender is used as the weight. Thus, the importance of witness information is proportional to the trustworthiness degree of the recommending agent.

5.2.2.3 Socio-cognitive trust models

The model proposed by Castelfranchi and Falcone is the prime trust model that explicitly stressed the importance of the socio-cognitive dimension of trust. They defined trust as a mental state based on which artificial agents can make delegation decisions within multi-agent systems. The authors consider trust as a combination of beliefs and goals that constitute this mental state. In a nutshell, an agent i trusts another agent j for doing the action α to achieve an objective φ iff [117]:

- i has the objective φ,

- i believes that j is capable of doing α,
- i believes that j has the power to achieve φ by doing α,
- i believes that j intends to do α.

Recently, this model has been extended and formalized in the context of the *ForTrust* project [93, 96]. This new model has been used in the context of detecting malicious (e.g. vandalism) contribution to Wikipedia [118].

5.2.3 Trust in cloud computing

Trust has been one of the main obstacles to the quick adoption of cloud solutions. To that aim, the significance of trust has been rapidly recognized among researchers and practitioners, making *trust management* a priority for the *cloud community* [143]. Several solutions have been proposed to *collect*, *manage*, and *propagate* evidences/indicators based on which trust relationships can be established. However, most of these systems rely on the adaptation of existing models which justifies the review we provided in this Section.

Recently, we identified a new category of works that tries to propose new models that addresses cloud specific challenges [88, 143, 50]. Theses models can be split into *Predictions-based Trust Models* and *SLA-based Trust Models*. We refer the reader to Deliverable 1.2 in which we provided a short review of these models.

Chapter 6 A Short Survey of Security Self-Management

We review the existing literature in the field of security self-management. In particular we break it down to the following categories:

- *Infrastructure Monitoring:* A security management system needs to know the current state and configuration of the infrastructure, in order to make security decisions.
- *Configuration and Deployment Policies:* Complementary to the access control policies of Chapter 5, we need to be able to specify the desired security and configuration state on the infrastructure level.
- *Security Properties:* We focus on the following security properties that the system should monitor, verify, or enforce.
  - *Tenant Isolation:* Virtualized and Cloud infrastructures are typically multi-tenant, i.e., physical resources are shared by multiple — potentially conflicting — users. It is necessary that the security management system ensures strong isolation among the tenants and their data. We review existing work complementary to the isolation architectures of Chapter 4, which focuses on hypervisor isolation and side-channel attacks. We focus here on isolation in the infrastructure at large and on information flow control.
  - *Integrity:* The integrity of the infrastructure components, such as the hypervisor, the infrastructure configuration, and the user applications is fundamental for the security of the system. We focus on work for the monitoring and enforcement of infrastructure integrity.
- *Dynamic Infrastructures:* Virtualized and Cloud infrastructures are very dynamic systems that are changed constantly. We review approaches and systems for the planning of configuration changes in such environments as well as the monitoring and analysis of changes.

6.1 Virtualization and cloud infrastructure monitoring

A tendency in virtualized data centers is that the complexity in the physical network shifts into the virtual one. The physical network configuration is simplified and its topology flattened, whereas complex virtual networks are created. Approaches for the discovery of the virtual network topology have been proposed [21, 142]. In public infrastructure clouds, the configuration of the firewall setup has been extracted and modeled as a graph [40]. The placement of VMs has been analyzed in public infrastructure clouds through network probing and co-location verification [157, 92]. Inspecting the configuration inside a virtual machine can be performed during runtime using virtual machine introspection [150, 136] or statically by analyzing the VM images [193]. An application and service centric graph model has been proposed for cloud infrastructures [31] with an automated discovery system [30]. Virtualization leads to this dynamic behavior including for compute and storage resources. We need to be able to monitor and model changes for network, compute, and storage resources in virtualized infrastructures. On the hypervisor level, Cloud Verifier [169] detects changes in the integrity of the hosted VMs on behalf of cloud customers. The system can also be used by providers to monitor the

integrity of their cloud platform. Cloud Radar [39] is a system that monitors virtualized infrastructures and analyzes those changes with regard to infrastructure security policies. From a performance point of view, vQuery [172] is a system for monitoring VMware environments that stores partial topological information in a graph model and tracks many performance metrics. In public infrastructure clouds, Amazon recently introduced a service called AWS Config [8] that provides change events to customers for their virtual environments. However, it provides no insights on changes in the underlying infrastructure, e.g., on VM migration and placement.

6.2 Policies for virtualization and cloud infrastructures

The security specification of VMs can be achieved with the concept of *Virtual Machine Contracts* [133], which are a policy specifications based on OVF that govern the security requirements of a virtual machine, e.g., to specify firewall rules. Similar to OVF, the objective of this language is linked to provisioning rather than expressing high-level security goals on the topological level. On the hypervisor level, *sHype* [161] is an implementation of access and isolation control for virtual machines, which uses a XML-based access control policy[1]. Again, the policy only applies to one entity in the virtualized system, i.e., the hypervisor hosting virtual machines. Xenon [134] provides XML-based policies for inter-VM communication, MAC with VM labels to implement Chinese wall, type enforcement, and time-based rules, as well as policies on hypercall and resource usage. VALID [35] is a language to express isolation and deployment policies for virtualized infrastructures. The language is based on the Intermediate Format [12] which is consumed by a wide range of model checkers and theorem provers for automated verification.

On the network router and firewall level, Bartal et al. propose a model definition language [19] to describe security zones, network topology, and firewalling. Hinrichs proposed a policy framework [94] that expresses policies as conditions leading to actions. The conditions can test on packet headers as well as global conditions, such as time and network load, which however require access to such live-data through the policy framework. Furthermore, the conditions can also operate on extended state associated with network flows, e.g., the association of users with source IPs. Actions include filtering (permit, deny), crypto requirements (encrypt traffic), and QoS. CloudPolice [153] is a distributed firewall with policies on tenant isolation, inter-tenant communication, and QoS (fair sharing, rate limiting). The policy format resembles the policies of Hinrichs [94] where conditions lead to actions. A condition is a logical expression of predicates on properties such as sender/receiver, packet header fields, time, and past traffic state. And action can allow, block, and rate limit matched flows.

Motivated by security in the SDN (Software-Based Networking) area, the Flow-based Security Language [95] allows the specification of high-level allowed network flows. *Alloy* [101] is a first-order logic modeling language, which is used, among other things, in network infrastructure modeling and analysis [139, 140]. Alloy can express structural properties as relations between objects as well as temporal aspects as dynamic models with states and transitions.

Data-centered policies have been proposed in the context of cloud computing and virtualized infrastructures. Santos et al. [166] proposes policy-based sealing of data, i.e., encrypted the data under an associated state and certain properties of the state (cf. [160]). The policy language is not fully defined, but the examples indicate logical expressions of equality and inequalities on state property key-value pairs. For example, the hypervisor must be equal to *CloudVisor* [204] and the version greater or equal to 1.

6.3 Information flow control and infrastructure isolation

Bacon et al. provide an overview and discussion on the usage of information flow control in secure cloud computing [16]. They conclude that DIFC is particularly suited for Platform as a Service (PaaS),

[1] Xen User Manual, Section 10.3.

because they believe this layer is best suited to provide labeling information on data and that existing open source implementations can be augmented. In fact, many approaches for information flow control on the PaaS level have been proposed. SilverLine [138] offers data and network isolation based on data labeling, however requires changes to both Xen and the guest VM kernel. SilverLining [110] focuses on Java application in the context of Hadoop map-reduce jobs and offers IFC through aspect oriented programming. It uses an information flow graph that captures users and files as nodes with read/write or disallowed flow directed edges. CloudSafetyNet [154] monitors information flow through HTTP tagging on the client side and socket monitoring. CloudFence [149] offers data flow tracking on the byte level using binary instrumentation. They claim to be more fine-grained than previous approaches that operate on the process level, such as SilverLine. CloudFlow [17] is based on VM introspection to monitor and extract the tasks with their SELinux security labels that running inside a VM. They implement a Chinese wall policy to prevent, for instance, that a VM with unlabeled tasks is running on the same physical server as a VM that runs a top secret task.
IFC also finds application in infrastructure management and administration. For instance the Chinese wall policy is implemented for administrators that log into customer virtual machines, in order to prevent an administrator to log into VMs of conflicting customers [198]. Further, *H-one* [81] is a system that uses information flow tracking to establish an audit log of VM configuration tampering by administrators.
A number of approaches also deploy the infrastructure in such a way that it provides tenant isolation. The Trusted Virtual Datacenter (TVDc) [26] offers automated deployment with isolation and integrity by leveraging a trustworthy hypervisor, trusted computing, and automated setup of network isolation [48]. The system uses Trusted Virtual Domains (TVDs) [49] to group resources together and defines an information flow policy, which is a similar concept to IFC. A formal isolation model for TVDc is presented in [29]. On the network level, CloudPolice [153] provides network access control in the hypervisor and essentially implements a distributed firewall. Tenant isolation may be also achieved using specific network architectures allowing integration with the customers on-site network [91, 197].
IFC approaches do not protect against side channel attacks, which recently have been demonstrated in PaaS [206], and additional security approaches are required. In particular, the enforcing of cache isolation [156] is important as many side channels rely on the shared cache, as well as to provide a VM placement algorithm that offers co-location resistance [15].

6.4 Infrastructure integrity enforcement and verification

Terra [82] was one of the first virtualization platforms including the TPM to provide integrity for VMs. In particular, a trusted hypervisor provides confidentiality and integrity to "black-box" VMs, and enables remote attestation using the TPM. However, it only provided load-time attestation. Overcoming the limitations of load-time integrity, the HIMA [13] system provides run-time integrity monitoring for VMs using a hypervisor-based measurement agent.
A variety of approaches have been proposed that enable integrity monitoring and verification not only for VMs but also for the hypervisor. For instance, HyperSentry [14] uses the System Management Mode (SMM), which operates on a more privileged level than the hypervisor, to perform integrity measurements. The measurements are triggered through an out-of-band channel using IPMI. HyperSafe [192] provides self-protection for the hypervisor and focuses on control-flow integrity (CFI). Finally, CloudVisor [204] uses nested virtualization to introduce a higher privileged layer under the hypervisor for the integrity protection.
In addition, MAC policies implemented by the hypervisor, such as sHype [161], have been analyzed with regard to integrity [158]. They model SELinux policies as an information flow graph and try to find violating paths.
Moving towards integrity verification and measurements of an entire virtualized infrastructure and not only individual hypervisors or VMs. The TCCP [165] system consists of a trusted hypervisor and a trusted coordinator. Similar to Terra, the trusted hypervisor provides integrity and confidentiality for

VMs. The trusted coordinator manages a set of trusted nodes based on their attestation results. The coordinator creates and migrates VMs only on the set of trusted nodes. Similarly, Krautheim [116] proposes a security architecture with dedicated measurement VMs that uses virtual TPMs [25]. The CloudVerifier [169] architecture enables end-users to verify cloud infrastructures. The system combines multiple existing components and approaches such as the Integrity Verification Proxy (IVP) [170], which performs remote attestation and VM integrity measurements, as well as trust anchorss [168] that establishes transitive trust in the cloud infrastructure.

6.5 Dynamic infrastructures: change planning and analysis

Misconfigurations in networks have been a problem in the operation of IT environments for a long time and solutions have been proposed. Mahajan et al. [129] studied misconfigurations in BGP routing configuration changes by listening to changes and assessing them. Kim et al. [114] analyzed the evolution of network configurations by mining a repository of network configuration files.
With the rise of software-defined networking, real-time monitoring and policy checking have been achieved in these environments [108, 111]. CloudWatcher [173] is a system that enforces mediation of network flows by policy. The administrator defines a set of security network devices with their capabilities. The security policy, called the security language interface, defines a flow condition and a set of security devices, which need to be traversed by the flow. They propose four algorithms to find an optimal routing path for a given flow that routes the flow over the required security devices. Bellessa et al. [23] proposes a system called NetODESSA that performs dynamic policy enforcement in virtual networks. A dynamic network policy governs not individual hosts, but is a high-level policy that operates on hosts with specific characteristics. The system monitors new flows and registers a decision by the reasoning engine based on the policy with the OpenFlow switches.

6.5.1 VM migration

Al-Haj and Al-Shaer [3] propose a framework for modeling and analysis of VM migrations. They model the migration as a constraint satisfaction problem, where for a given VM placement a migration sequence needs to be found that satisfies the desired VM placement and additional constraints. The constraints capture dependency, performance, and requirements, such as, the prevention of co-location of conflicting VMs. They use a SMT solver to find a sequence of migration steps that satisfy the constraints.
Cloud Calculus [102] is a formal framework for the modeling and analysis of VM deployments, in particular maintaining security invariants given by firewalls during VM migrations. The model is based on the mobile ambient calculus, a process calculus, that models VM mobility as well as firewall configuration changes. The preservation of policies during VM migration must ensure that packets are treated the same before and after migration. Follow-up works, which are extending Cloud Calculus, include the modeling and analysis of distributed firewalls as well as intrusion detection systems and VPNs. Jarraya et al. [103] extends the Cloud Calculus to cover distributed firewalls through composition of firewall configurations. Eghtesadi et al. [69] uses the Cloud Calculus approach to focus on the preservation of monitoring and tunneling configurations in IDS and VPN settings.
Focusing on operational problems as part of VM migration and configuration changes, CloudInsight [10] tracks VM configuration changes and correlates changes with performance problems. They model and monitor physical as well as virtual machine configurations through a polling agent on the hypervisor. If a problem has been reported for a VM, the system identifies suspected change events based on the instability of a change. The suspected events are ranked based on their sensitivity, i.e., how often an attribute changes, in a local (VM instance) and global view. The most likely root causes are then interactively remediated with the help of the user.

6.5.2 Change planning

Generalizing the analysis and planning of VM migration to changes in general, we now discuss approaches for network and virtualization infrastructure change planning. Change planning and the analysis of changes for network routers has been addressed by existing research [182, 5], focusing in particular on performance properties.

Hagen [90, 89] studies the verification of change operations in the context of statically and dynamically routed networks. They operate on objects and attributes of infrastructure components tracked in a CMDB. A change is modeled as a set of predicates and a set of effects on those attributes. A case study investigates the change of network routing from high-capacity to low-capacity network. They model change operations for taking down an interface and shifting traffic. A safety constraint specifies that to only route high-capacity traffic via high-capacity routers.

Bleikertz et al. [37] propose a system that analyzes virtualized infrastructure changes using an operation model. The model is based on graph transformations of operations on a graph model of a virtualized infrastructure. The system can intercept operations performed by administrators, analyze them, and decide to reject or accept the operations.

Kikuchi [113, 112] studies configuration synthesis and vulnerability analysis of dynamic virtualized and cloud infrastructures using formal methods. For the configuration synthesis, they model four operations: establish a physical connection, give access to another component, join a VLAN, and VM migration. The synthesis is further guided by constraints on physical and virtual network connectivity as well as host capacity. Given a goal condition, the Alloy Analyzer tries to find a sequence of operations that satisfy the constraints and reach the goal condition. The initial state of the system is translated from a configuration database (CMDB) into Alloy. With regard to operational vulnerability analysis, Kikuchi focuses on services availability, in particular due to single point of failures, over capacity, and load-balancer misconfigurations. They model changes to the infrastructure such as crash faults, VM migrations, and monitor changes of a high-availability load-balancer. As part of the state transitions they model the dependency among the components, for instance, if a physical server fails then all the VMs running on that server will also shut down. They employ NuSMV as the model checker with CTL policies.

Pearson argues for accountability in cloud computing [151] and highlights that tool-supported accountability is essential due to the automated and dynamic nature of infrastructure clouds. We need to automate to a large degree the monitoring and verification of dynamic virtualized infrastructures.

Chapter 7 Preliminary Architecture for Virtualization & Security Self-Management

In this chapter, we present a preliminary architecture for the SUPERCLOUD virtualization infrastructure and security self-management for computation. The architecture meets requirements identified in Chapter 2 and proposes new designs based on approaches and technologies reviewed in Chapters 3–6. We first present the design principles for the virtualization infrastructure and self-management architecture (Section 7.1). We then give an overview of the overall architecture (Section 7.2). We finally present the design of its different components: computation hypervisor (Section 7.3), isolation and trust management (Section 7.4), self-management (Section 7.5), configuration compliance (Section 7.6), and authorization (Section 7.7).

7.1 Design principles

7.1.1 Virtualization architecture

The user-centric, fully interoperable, multi-provider cloud still remains elusive. "Horizontal" multi-cloud interoperability limitations come on top of "vertical" multi-layer security concerns. Such issues may be addressed through the *virtualization architecture*.

Fulfilling the user control requirement: modular hypervisor / nested virtualization

Malicious administrators may leverage their extended privileges to snoop the private execution state of applications, without any possible user control. User-centric security allows the user to customize the protection of the virtualization layer to prevent such threats.
Unlike monolithic General-Purpose Hypervisors (GPH), *Component-Based Hypervisors* (CBH) open to users fine-grained control of their deployed resources. This is achieved by modularization of the traditional GPH control plane (e.g., Xen Dom0). This design requires the user to adapt his control framework to this new logic, breaking compatibility with legacy management toolkits.
Trade-off between provider and user control may also be achieved through layering, by separating user-level and provider-level hypervisors in a *Nested Virtualization* (NV) architecture [194].

Fulfilling the minimal TCB requirement: micro-hypervisor

Malicious co-located tenants may try to exploit flaws in the huge TCB of modern virtualization stacks to escape execution environment isolation. For this requirement, *Micro-Hypervisors* (MH) are clearly the best design alternative. A GPH presents a large TCB and is traditionally prone to failures due to the size of critical code. MHs reduce TCB through modularization, expelling in user-space device drivers and other system components.

Fulfilling the interoperability requirement: nested virtualization / containers

An important unsatisfied property of multi-clouds concerns the possibility to migrate execution environments across different providers as transparently as for single clouds. This lack of flexibility is mainly due to provider lock-ins and incompatible technological choices.

Several GPH-based NV architectures may provide a user-centric virtualization layer that could be executed over different providers to implement a multi-cloud abstraction layer. However, such solutions do not really present a minimal TCB architecture or address security and interoperability simultaneously without requiring more than two layers of virtualization.
A *container-based design* might represent a very good trade-off between legacy support, interoperability, near-optimal performance, also providing good scalability. However, security and control requirements are not completely satisfied.

Fulfilling the legacy support requirement: nested virtualization

To encourage its adoption, a multi-cloud architecture should require minimal porting effort in application and control logic adaptation. NV inherits the GPH transparency, allowing to control resources with the same interfaces. CBH and MH introduce a new control logic and/or hypervisor interface.

Design principles: a summary

A SUPERCLOUD virtualization architecture with a hybrid design could be a viable solution to achieve a reasonable trade-off between the key requirements for computation. In the following, we will thus present a new virtualization architecture combining NV, MH, and CBH. Leveraging NV interoperability and legacy support, the architecture provides to users a transparent federation of multiple-provider resources. It also features a MH including CBH-like modules as NV lower-layer hypervisor for a minimal TCB and to enable users to directly control hypervisor resource management components.

7.1.2 Self-management architecture

Distributed cloud infrastructures take complexity and threats to the next level compared to single clouds. Vertically, vulnerabilities across infrastructure layers add up to heterogeneity of defenses, which must now also be considered horizontally across cloud providers.

Mitigating security complexity: transparent self-protecting multi-cloud

Autonomic security management has been gaining momentum as simpler, faster, and more flexible approach to respond to threats. In a *self-protecting system*, security parameters are autonomously negotiated with the environment to match the ambient estimated risks to provide an optimal level of protection [58]. This is realized through *autonomic security loops* with stages of *detection* (monitor, capture security context), *decision* (elaborate reaction plan to come back to secure state), and *reaction* (enforce security response to eradicate the threat). This approach enables lighter administration, increased reactivity, lower security management costs, and graduated threat response.
SUPERCLOUD includes different infrastructure dimensions (multi-layer, multi-provider) and security services. A security response can thus only be elaborated through *orchestration* of *multiple autonomic security loops*. One needs to distinguish *vertical (cross-layer)* and *horizontal (multi-provider)* dimensions of orchestration (see Figure 7.1).
Security management may also be envisioned more broadly than pure detection and threat prevention. Other security services may be considered such as managing trust (e.g., proving, verifying trust assertions). Other non-functional services (e.g., energy efficiency) are also possible. Orchestration then amounts to coordination of per-security service autonomic security loops, possibly composed with other autonomous systems (e.g., for energy efficiency).

Reconciling security automation and exploding layer complexity: cross-layer self-protection

Vertical security orchestration aims to automate security within and across layers of the distributed cloud. The cross-layer approach is expected to improve security by helping to capture the overall extent of an attack spanning multiple layers and to better respond to it.

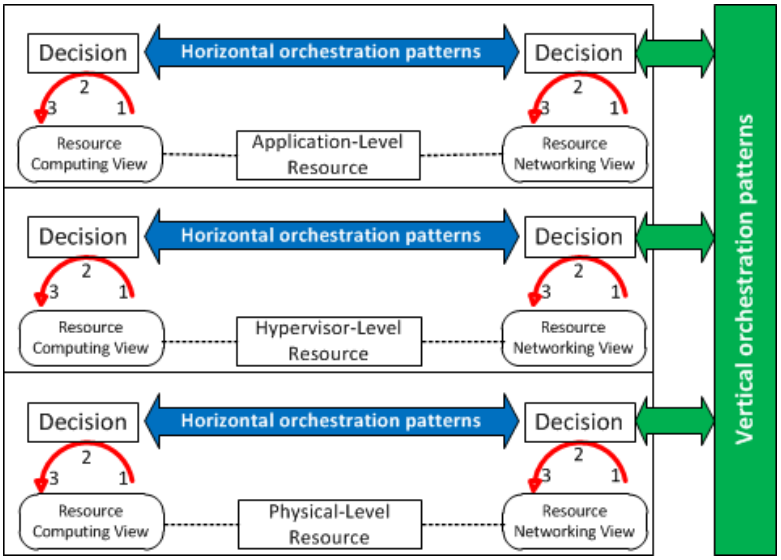


Figure 7.1: Horizontal and vertical loop orchestration [189]

- User control and security automation may apply to different levels of the infrastructure: VMs, containers, user-centric virtualization stack, user-controlled modules in the provider infrastructure... *Layer Orchestrators* (LO) are introduced to control security in each level. An LO manipulates security information gathered from probes in the infrastructure, and uses enforcement points to modify the layer security state. A *Vertical Orchestrator* (VO) achieves cross-layer self-protection through LO coordination.
- The extent of this control depends on the openness of the underlying provider infrastructure: deeper control in the layers of a private cloud (OpenStack) is more likely than in a public cloud (Amazon EC2). An extra component, the *Arbiter*, may be needed to capture and enforce trade-offs between user and provider control over security. This arbiter would typically steer multiple VOs: one handling user-controlled security (as previously), and another capturing provider-controlled security – the provider may already have its own security supervision system[1].

Reconciling security automation and provider heterogeneity: cross-provider self-protection

Horizontal orchestration aims to provide a homogeneous level of security across clouds. Security automation is considered independently from the underlying providers, e.g., to detect and mitigate threats propagating from provider to provider. Corresponding reaction policies may be distributed and enforced using security mechanisms of underlying providers. Control points to achieve such homogeneity may be the VOs defined previously (typically one per provider).

- A unique *Horizontal Orchestrator* (HO) may control the whole set of underlying provider clouds. This approach offers high user control, but low scalability. The HO is also a single point of failure and privileged target for attacks.
- HOs may also *collaborate* using different patterns [122]. Design alternatives are similar to those for network control for SDN architectures (see Deliverable D4.1). Peer-to-peer interconnection offers high scalability, but may require sophisticated protocols to achieve true interoperability and automation if the number of providers increases. The HO can also act as broker between VOs controlling a subset of the multi-cloud infrastructure (e.g., as OpenStack availability zones). This allows to negotiate the conditions of security interconnection between heterogeneous security

[1]The Arbiter could also steer other VO/LO-based autonomous management systems for other concerns than security (e.g., for energy efficiency).

management systems. Hierarchical HOs are also possible to distribute control. The latter option tends towards higher levels of optimality for the security response, but with greater complexity as the number of hierarchical levels increases.

- Horizontal orchestration may be centralized among all user-centric clouds (1 single HO for all U-Clouds), completely decentralized (1 HO per U-Cloud), or hybrid.

Once the security management architecture is thus defined, the last step is to embedded it into the virtualization architecture. This means placing detection, decision, and reaction components of autonomic security loops in the different virtualization layers and across providers to achieve true unified automated security management of the multi-cloud system.

7.2 Architecture high-level overview

7.2.1 Positioning in overall SUPERCLOUD architecture

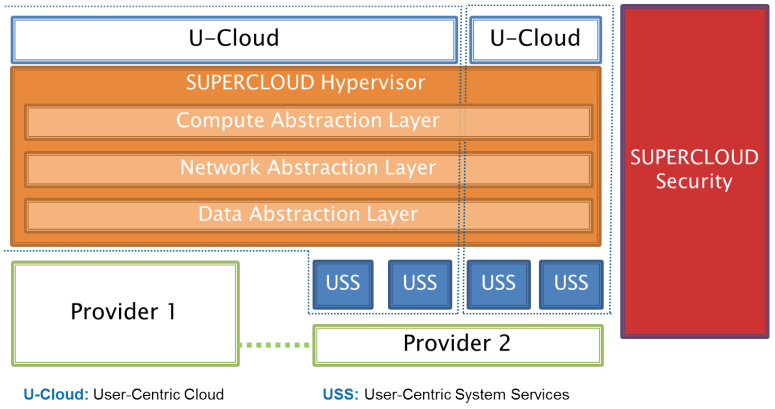


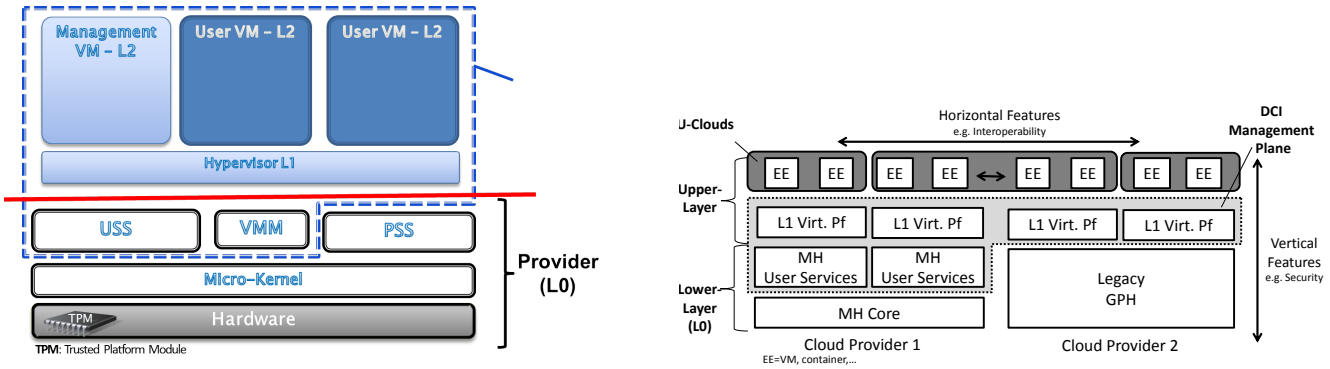
Figure 7.2: Relation between computing virtualization architecture and overall architecture

Figure 7.2 gives a high-level overview of the overall SUPERCLOUD architecture defined in D1.1. SUPERCLOUD considers multi-provider cloud infrastructures – typically a federation of *public clouds* (Provider 1 in Figure, e.g., Amazon EC2) with low openness and *private clouds* (Provider 2 in Figure, e.g., OpenStack) with higher openness enabling greater user control over the infrastructure.

- *User-centric clouds (U-Clouds)* run above providers in a provider-independent manner.
- A *SUPERCLOUD hypervisor*. This should be understood as a conceptual distributed virtualization infrastructure for running U-Clouds above compute, data, and network resources of the different providers. The SUPERCLOUD hypervisor is viewed as formed of *several abstraction layers* for compute, data, and network resources (represented on the Figure as stacked layers, but without any hierarchical relationship).
- Security services (captured as *SUPERCLOUD security*) guarantee protection of U-Clouds, e.g., self-management of security. U-Cloud protection may be customized thanks to *User-centric System Services (USS)* that extend user-control of the U-Cloud from the SUPERCLOUD hypervisor into the lower layers of the provider infrastructure (private cloud case).

The virtualization and self-management architecture for computing resources specified in D2.1 aims to define: (1) the structure of the compute abstraction layer; and (2) the corresponding security services for computation in the SUPERCLOUD security component. The next two sections provide respectively high-level overviews of the computing virtualization architecture alone (Section 7.2.2), and including security self-management features (Section 7.2.3).

7.2.2 Virtualization architecture



(a) (b)

Figure 7.3: Computing virtualization architecture: (a) single provider; (b) multiple providers

Figure 7.3a gives a high-level overview of the virtualization architecture without self-management features for a single provider. U-Cloud implementation relies on nested virtualization which offers interesting benefits in terms of both interoperability and security to guarantee VM protection in spite of untrusted virtualization layers. Two layers may be distinguished: the *Lower-Layer* (or L0) controlled by the provider, and the *Upper-Layer* (or L1) controlled by users.

- In L1, the user can run *Execution Environments (EE)* which may be VMs (named L2 VMs), or containers – in which case the L1 virtualization platform is a container-platform, Docker style. L2 VMs may be end-user VMs (hosting applications) or management services (e.g., appliances, L1 Dom0). Multiple L1 instances may be deployed across providers (see Figure 7.3b), e.g., to migrate EEs transparently. This requires interfacing with the WP4 network architecture.
- L0 contains the provider hypervisor and its services. The type of hypervisor may depend on openness of the provider. L0 may be a GPH for a public cloud (e.g., Xen for Amazon). For a private cloud, more modular forms of hypervisors such as a MH may be preferred. This enables clean separation of *provider-controlled services* (PSS) and of a part of L0 under user control. Typically: a *VMM component* for core virtualization functions, e.g., L1 VM creation/deletion; and *infrastructure services* (USS), e.g., checkpointing/recovery, live migration, etc.

7.2.3 Virtualization and self-management architecture

Overview

Figure 7.4 shows a high-level overview of the virtualization for computation including self-management features. Two parts should be distinguished:

- The *virtualization infrastructure* provides a distributed abstraction layer for computing resources spanning multiple cloud providers. The general system design was briefly presented previously and will be explained in more detail in Section 7.3.
- The *self-management infrastructure* implements autonomic security management for the distributed cloud (see Section 7.1.2). Security self-management is addressed both:
  - *Vertically:* self-management is realized through a hierarchy of LOs controlling security in each layer: VM/container, L1, and L0. LOs are coordinated by an overall security manager in charge of supervising cross-layer security management. A similar provider-controlled security supervision framework aimed at guaranteeing the protection of its own infrastructure is represented here for simplicity. Such a design is provider-specific, and other

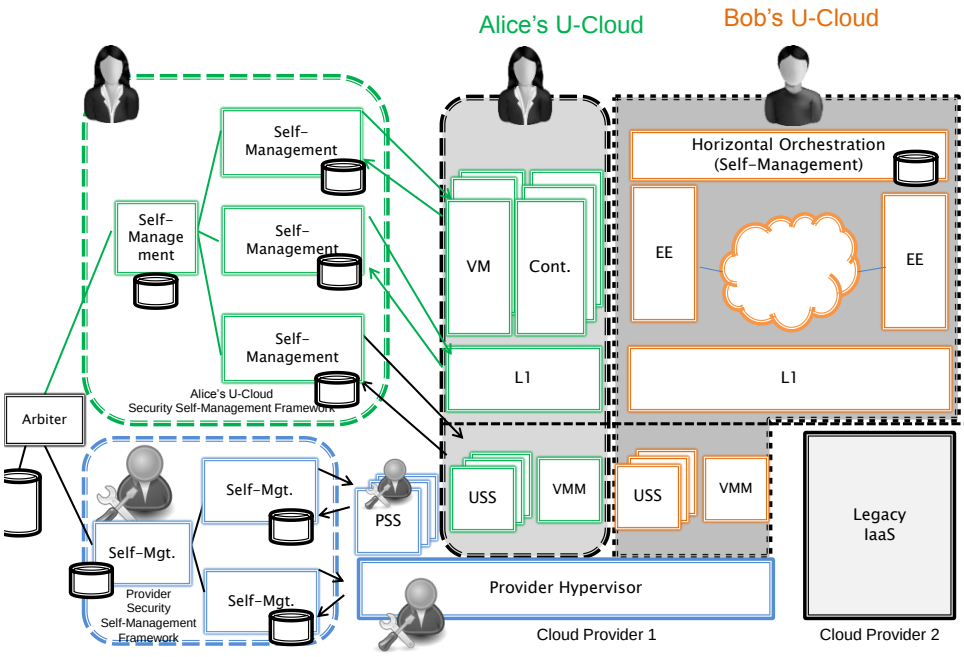


Figure 7.4: Overview of virtualization architecture with self-management features

designs may be possible. An *arbiter* component manages trade-offs between provider and user control over security, such as negotiation over SLAs and security policies. In the design shown, each such hierarchy is U-Cloud-specific to customize self-management per group of customers forming the U-Cloud. Self-management could also be shared among U-Clouds.

– *Horizontally:* self-management is handled by an horizontal orchestration framework, with several possible designs (see Section 7.1.2).

This architecture needs to be enhanced with a number of features:

- *Isolation and trust*: different types of *isolation technologies* should be supported, e.g., relying on hardware security mechanisms; *trust management* should also be explored vertically across infrastructure layers, horizontally across providers, and across users. The corresponding components will be presented in Section 7.4.
- *Configuration compliance*: this service is needed for auditability, towards regulatory authorities, or towards customers to increase the level of trust on the security hygiene of the distributed cloud. This component may typically be implemented as an USS to have an overall view of the U-Cloud, independently from provider claims. Its design will be presented in Section 7.6.
- *Authorization*: one authorization component is needed to perform resource access control at different levels of the infrastructure. Its design will be explained in Section 7.7.
- *Policies and orchestration*: more details are needed regarding *user-centric* definition of *security policies*, relation between vertical and horizontal self-management, and *orchestration* of different security services. Those elements will be explained in Section 7.5.

Self-management and authorization

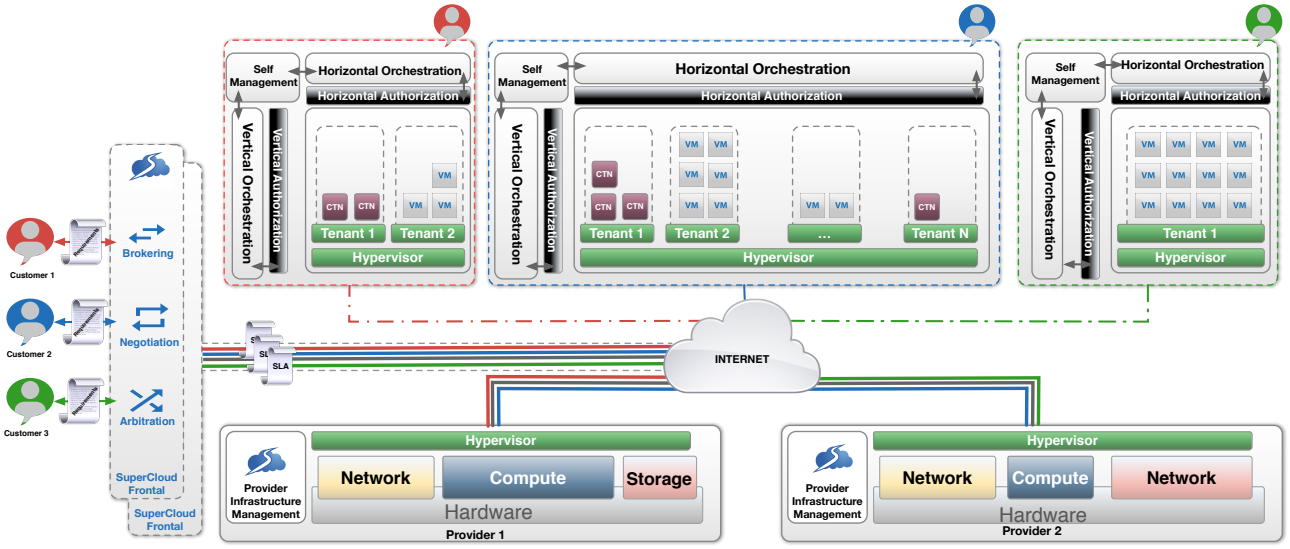


Figure 7.5: Relation between self-management and authorization components

Regarding self-management, additional components are needed (see Figure 7.5) such as a SUPER-CLOUD *front-end* (or Frontal) in charge of capturing customer security requirements. We envision also *Brokering*, *Negotiation* and *Arbitration* capabilities that can be embodied in this *Frontal* in a centralized approach (as depicted in the Figure 7.5), or scattered among several dedicate components in a distributed view.

- ***Brokering*** encapsulates the *discovery* and *advertisement* of *cloud services*. The *Broker* is also responsible of reservingthe necessary resources with one or many providers in other to fulfil the customer requirements.
- ***Negotiation*** tries to reduce the gap between customers requirements and providers constraints. The solution proposed after negotiation should reflect the optimal configuration for both parties.
- ***Arbitration*** aims at solving conflicts that may arise between Customers and Providers.

U-Cloud deployment over a provider infrastructure is managed by the *Self-Management Component*. We focus here on its role to manage *access control* decisions.

In the previous Figure, we consider using two *authorization* management mechanisms. The *Horizontal authorization manager* is in charge of making *access control* decisions across *tenants*, *users* and *providers*. The *Vertical authorization manager* manages *access control* decisions across layers. In some cases, horizontal and vertical authorization decisions may be in conflict. In such events, we rely on the *Self-Management Orchestrator* to resolve such conflicts and arbitrate between both decisions[2].

[2]As the joint use of MotOrBAC and OrBAC API (cf. Section 7.5.1) can detect and resolve conflicts within authorization policies, its use for this purpose requires few effort and minor adaptations. This framework and other authorization frameworks will be presented in Section 7.7.

7.3 Computation hypervisor

7.3.1 High-level design

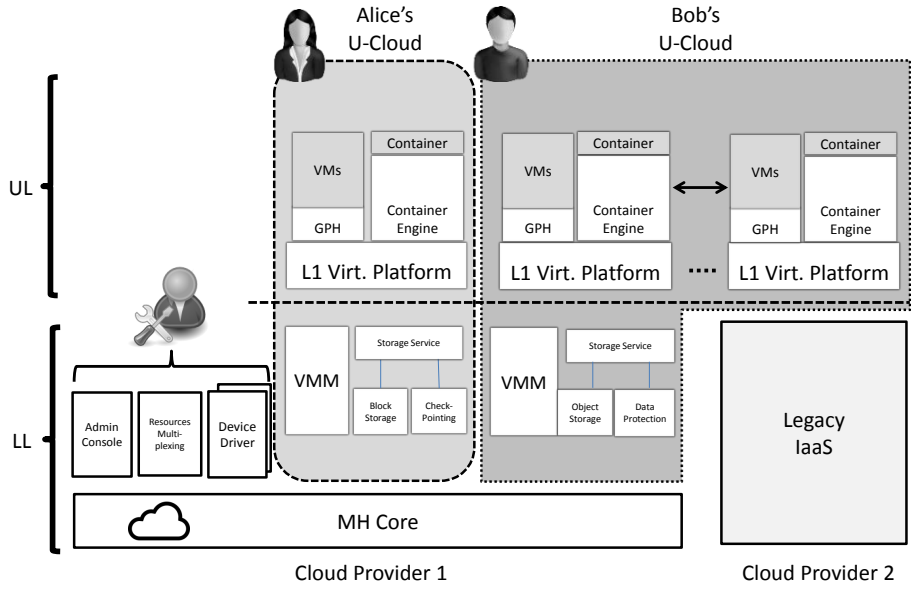


Figure 7.6: Computation hypervisor system design

There are two main alternatives to service provider VM protection: (1) direct security control by the customer; and (2) protection by the cloud provider. In (1), security is managed through customer-controlled appliances or intra-VM mechanisms, but remain with very limited privileges. In (2), protection is achieved through provider-controlled appliances or hypervisor-level mechanisms. The objective of the architecture of the hypervisor component is to overcome flexibility limitations that tie infrastructure services to the provider, causing vendor lock-in for the customer.
For the computation hypervisor, we adopt a design that is a combination of micro-hypervisor (MH), component-based hypervisor (CBH), and nested virtualization (NV) architecture. The general system design of the hypervisor is shown in Figure 7.6.
We consider a NV-based design to support U-Clouds [195]. The architecture is composed of two independent virtualization layers: the *Lower Layer (LL)* (L0 virtualization layer) and the *Upper Layer (UL)* (L1 virtualization layer):

- The LL is composed of a MH under provider control and of a set of user-land infrastructure services for increased user control over U-Clouds. Leveraging the strict modular architecture of MH, the TCB is now reduced and some of most failure-prone components (e.g., device drivers) are now outside the architecture core.
- The UL federates cloud resources in a provider-independent manner. It implements interconnection between different providers. The UL also enables to realize the U-Clouds, ranging from lightweight and less isolated EEs (e.g. Linux Containers) to typical IaaS EEs (e.g. hardware-assisted VMs). It provides the means for full U-Cloud SLA customization, notably through configuration of the required LL-level user-infrastructure services.

7.3.2 LL design

The general LL design is hybrid between minimizing the virtualization layer (MH à la NOVA [177]) and control disaggregation with several user domains (CBH à la SSC [47]). We assume the provider-layer to be MH-based. This assumption may hold for an open cloud provider architecture. This design choice provides a solid foundation to meet all vertical features, notably smaller TCB size and enhanced

global security such as guaranteeing VM security even if intermediate layers are compromised. This assumption does not seem unreasonable for upcoming years as:

1. Despite most current IaaS platforms still being GPH-based, there is a strong trend towards making the hypervisor more minimal and flexible, as witnessed by disaggregation of hypervisors of the mainstream IaaS platforms;

2. MHs themselves are becoming component-based, with possibility of interoperability to avoid any further risk of IaaS lock-in as shown by first multi-MH OSes.

The *MH core layer* provides a tiny set of key kernel features, e.g., scheduling, MMU management, IPCs, handling VMX-related events and interrupts. Following CBH principles, each user controls a *VMM* and a set of *user-space services*. The VMM is a lightweight implementation of the L0 hypervisor logic, with the same reliability and security benefits as in NOVA. The VMM is also crafted to enable NV through support for running nested VMX instructions. User-space services enable users to customize infrastructure resource management for their U-Clouds. The cloud system administrator directly controls resource management policies, system-wide resource multiplexing, and device drivers, but without control over the MH core.

- *User standpoint:* the LL architecture increases control over infrastructure, adding incrementally new infrastructure management services under the hood without disrupting legacy applications.

- *Provider standpoint:* security is strongly enhanced due to two-level isolation by the MH core and L0 hypervisor virtualization. Integrity of user-controlled L0 system services can be guaranteed through hardware security mechanisms (e.g., Intel TXT/SGX technologies).

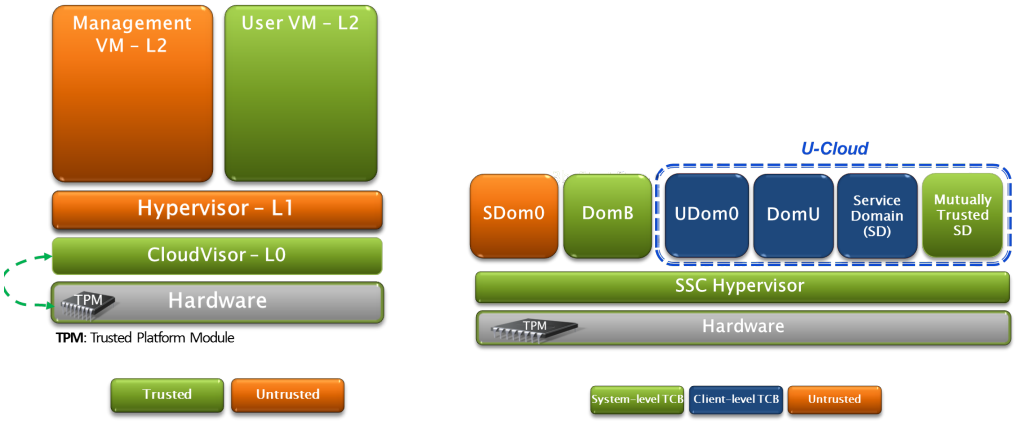
7.3.3 UL design

The general UL design does not export hardware resources with a single interface, but with a spectrum of interfaces of varying abstraction levels. Indeed, the UL should allow building U-Clouds where VM, network, and storage SLAs may be personalized independently from the underlying provider. Such U-Clouds may be defined at the PaaS or IaaS levels. We consider a L1 virtualization platform enabling to support both VMs and containers, and thus having a flexible virtualization interface, ranging from hypervisor to OS-level virtualization on which such EEs may run [185].
Extending the Library OS philosophy that adapts the OS to the application [115], the L1 platform should adapt the virtualization technology to applications according to different trade-offs. The UL should thus realize the widest possible spectrum for supported L1 virtualization techniques. Eventually, the UL will support dynamic on-demand provisioning of virtual execution environments with their GPH/container-engine.
The L1-virtualization platform is GPH-based to guarantee interoperability and other horizontal features through an inter-cloud blanket layer [195].

7.3.4 Relation to isolation technologies

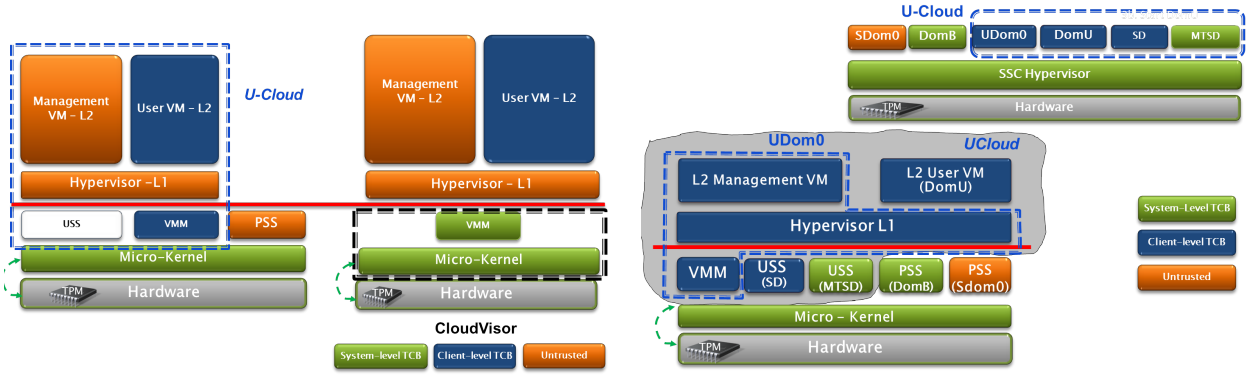
The virtualization architecture may support several types of isolation technologies (see Section 7.4):

- *Single provider*: the architecture may be mapped to both CloudVisor (see Figure 7.7a) and Self-Service Cloud (SSC) isolation architectures (see Figure 7.7b), notably in terms of trust model. Figure 7.8a and Figure 7.8b show how the virtualization architecture allows to express those designs. Figure 7.9 summarizes where the SUPERCLOUD architecture stands w.r.t. flexibility and security compared to existing designs such as CloudVisor (security only for single clouds), XenBlanket (interoperability for multiple clouds with no security), and SSC (security and user-control but for a single provider). Further work is needed to see the SUPERCLOUD virtualization architecture can express other isolation technologies such as TEE/hardware mechanisms (e.g., Intel SGX) or purely cryptographic techniques.



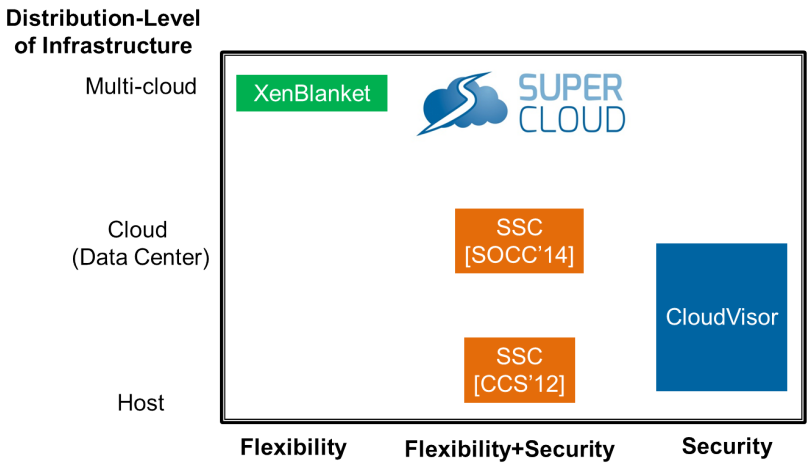
(a) (b)

Figure 7.7: Isolation architectures: (a) CloudVisor; (b) SSC



(a) (b)

Figure 7.8: Mapping SUPERCLOUD virtualization architecture to: (a) CloudVisor; (b) SSC



- **Flexibility:** ability to deploy (security) infrastructures services according to customer needs to realize a SSC.
- **Security:** protection of user VM against malicious cloud administrator (but cloud provider being trusted).

Figure 7.9: SUPERCLOUD virtualization architecture vs. other virtualization designs

- *Multiple providers:* relation with SDN-based network isolation, as studied in WP4, may also be established. Further refinement of this preliminary architecture is needed to specify the link between the L1 hypervisor components forming each side of a distributed U-Cloud and the network hypervisor and SDN controller components of the WP4 network architecture that notably manage network-wide isolation between tenants.

7.4 Isolation and trust management

Components managing isolation and trust can be divided into a "software" trust management framework and a "hardware" trust management framework. Section 7.4.1 describes a preliminary trust management management architecture. Sections 7.4.2 and 7.4.3 then present respectively "software" and "hardware" trust management frameworks.

7.4.1 Trust management architecture

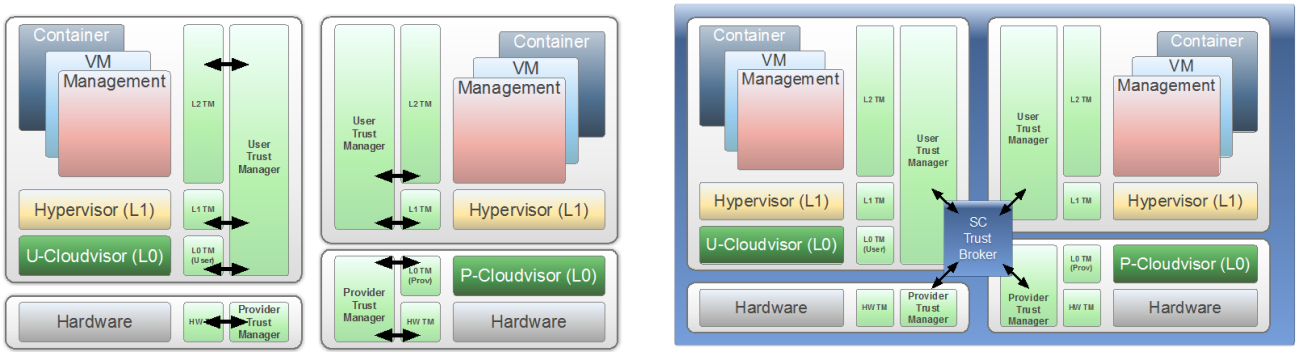


Figure 7.10: Trust management architecture

A preliminary trust management architecture is shown in Figure 7.10. *Chains of trust (CoT)* are established between components of the infrastructure. Each element in the CoT vouches for the trustworthiness of the next. This approach is advocated by the Trusted Computing Group (TCG). Integrity of a component may be verified by following the CoT to a *Root of Trust (RoT)*, usually protected by tamper-proof hardware such as a TPM.

- Vertically, the distributed cloud infrastructure is modeled as several layers, software and hardware containing resources. The *physical layer* consists of computing (CPU, memory) and storage hardware resources. The *virtual layer* contains the VMs (virtual CPU and memory) and virtual storage. The *application layer* leverages virtualized resources to run applications. One or several *virtualization layers* manage allocation of host resources among VM instances.

- Horizontally, the infrastructure is seen as a federation of *provider domains* that manage resources within a given perimeter, and according to a common policy. To simplify, we assume domains are layer-specific. *Domain federations* group together domains in each layer. In reality, the infrastructure is two-dimensional, with cross-cutting layers and domains.

The figure shows how, similarly to self-management, a hierarchy of trust managers may be defined in each layer of the virtualization architecture to build CoT vertically across layers. Trust managers may either be user-related or provider-related. A central SUPERCLOUD trust broker is also defined to manage multi-provider trust relationships.

7.4.2 "Software" trust management framework

The "soft" aspect of trust aims at managing trust relationships across users and across providers. This task is achieved by the Trust Manager embodied in the self-management architecture and illustrated hereafter.

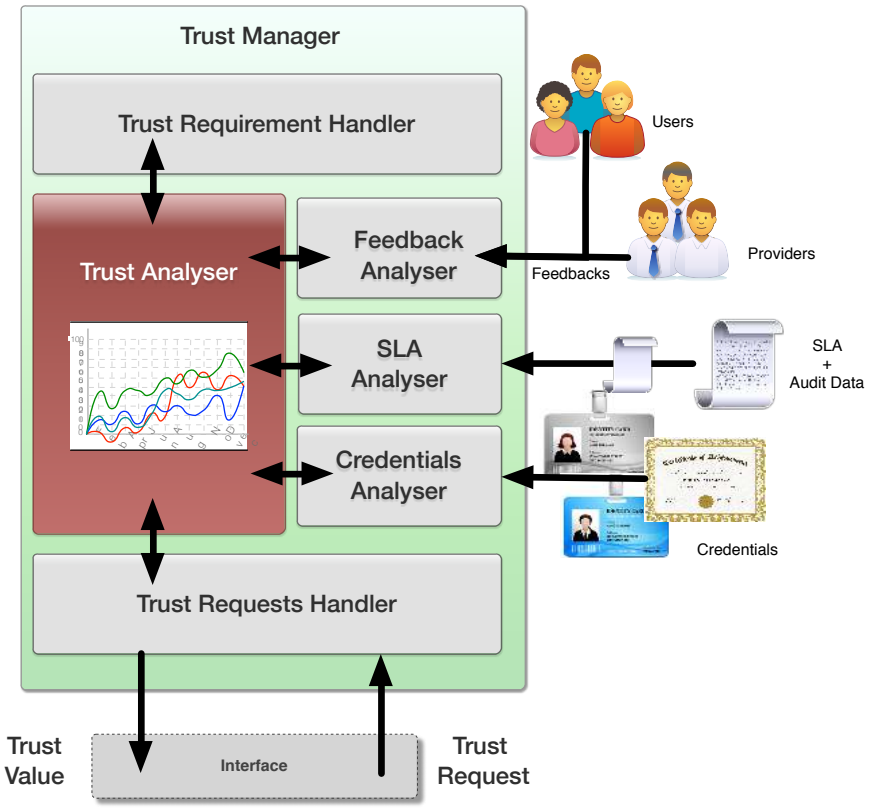


Figure 7.11: Trust manager overview

As depicted in the Figure 7.11, the Trust Manager is built upon six components that we briefly present hereafter.

The *Feedback Analyzer* is responsible of the collection and analysis of feedbacks and opinions. The *SLA analyzer* is responsible of extracting and evaluating SLA metrics. The *Credentials Analyzer* builds chains of trust and evaluates the validity of credentials. The *Trust Requirements Analyzer* parses and extracts trust requirements. The *Trust Analyzer* encapsulate the schemes used to compute trust. Finally, the *Trust Requests Manager* orchestrates and coordinates the collaboration of the aforementioned components.

A more detailed description of each of the above components can be found in deliverable D1.2.

7.4.3 "Hardware" trust management framework

The "hardware" trust management framework relies on components for managing *isolation* (i.e., isolation technologies), for handling *cross-layer trust*, and on *hardware security mechanisms* (i.e., to establish RoT), as shown in Figure 7.12.

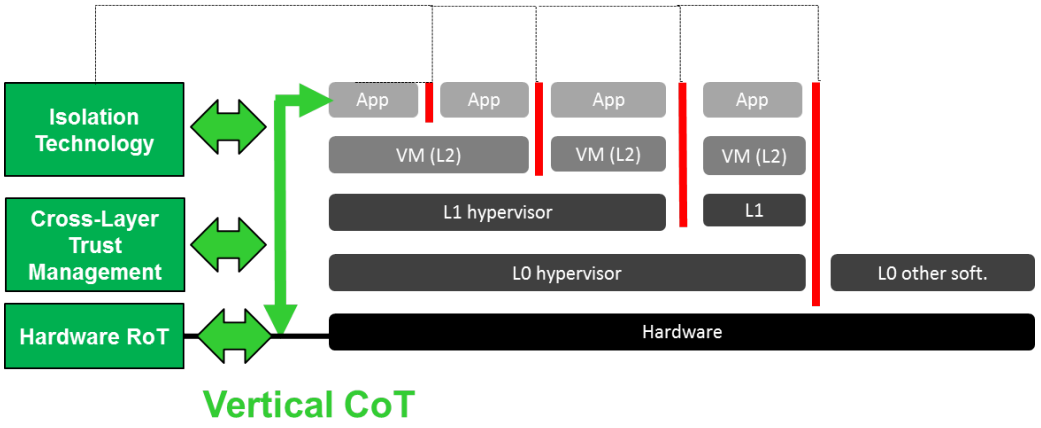


Figure 7.12: Hardware trust management components

7.4.3.1 Isolation technologies

As mentioned before, the hypervisor and virtualization architecture could be adapted to support both CloudVisor and SSC virtualization architectures (see oudVisor is a nested virtualization architecture that is designed to protect security and privacy of the clients from the administrative domain. Because the CloudVisor TCB is very small, it has a smaller attack surface. It also uses cryptography to ensure security and privacy of client VMs. Moreover, it provides protection to virtual disks owned by a VM through I/O encryption. Compared to CloudVisor, SSC provides more flexibility to clients to control their VMs. Additionally, it does not rely on nested virtualization and therefore has less overhead on client VMs. Finally, by using MTSDs the cloud provider and clients can execute mutually-trusted services for regulatory compliance.
To give clients more control over their cryptographic operations, the CaaS architecture can also be adapted. In CaaS, clients can establish and control Cryptography-as-a-Service in the cloud where they can securely provision keys, and even implement a private security module such as Virtual Hardware Security Module (vHSM) or smartcard. A client-specific secure execution domain protects the execution of all cryptographic operations. CaaS will also provide a protection for legacy VMs that are not adapted to this architecture.
Figure 7.13 shows the adaption of CaaS in the overall architecture. Domain management tasks are moved to a new Trust Manager. The Trust Manager is privileged to build new domains, and is chained to the TPM to provide trust inside the module with capabilities such as machine hardware encryption, signing, secure key storage, and attestation. Provider Security Services (PSS) are then degraded to an untrusted domain while keeping its purpose as administrative domain. The PSS have access to the hardware but are not able to access the memory of the Trust Manager. Within each U-Cloud, cryptographic operations and primitives are encapsulated into a separate client specific *Secure Domain (SD)* to isolate them from the vulnerable client VMs. The SD is deployed so that it will prevent internal and external adversaries from accessing the U-Cloud secrets. This protection is integrated in the entire VM life-cycle. Through SDs, U-Clouds interact with virtual TPM (vTPM) instances that are implemented in Trust Manager. The keys of this vTPM instance are bound to the hardware TPM. The SD is within the same level as the L1 hypervisor and acts as a transparent secure device proxy layer between each U-Cloud and the Trust Manager. This layer can be used to for example booting a fully encrypted VM image.

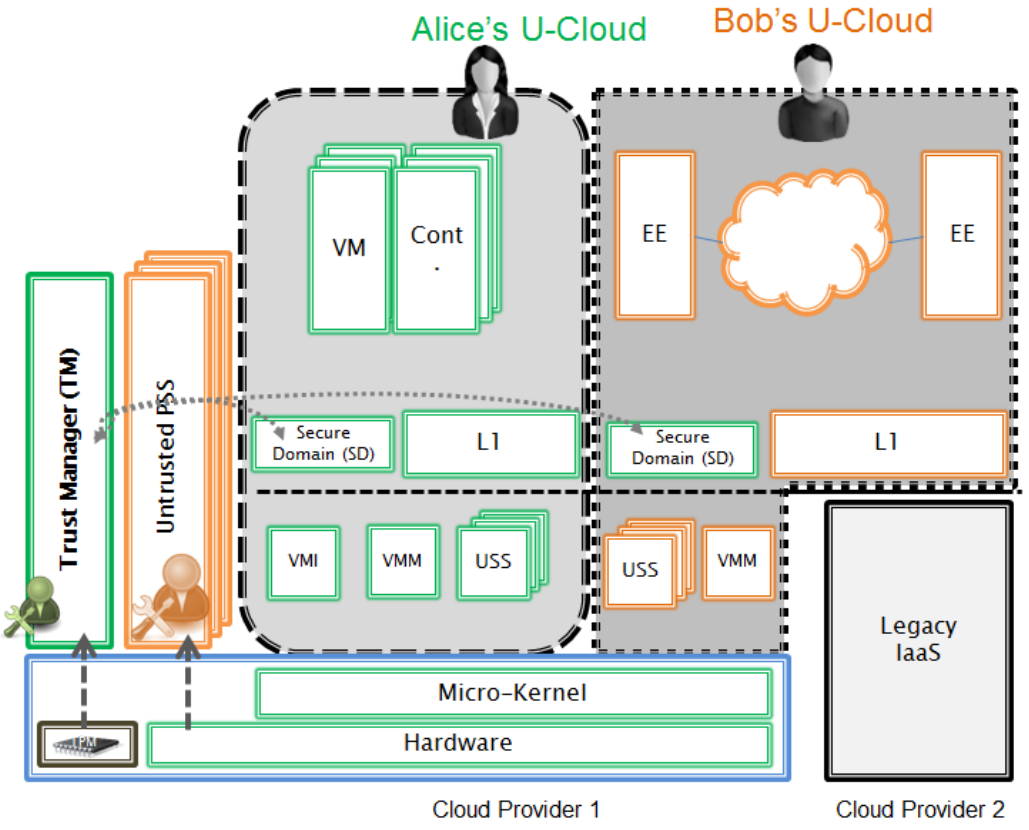


Figure 7.13: Isolation technology: adaptation of CaaS and SSC in the architecture

7.4.3.2 Cross-layer trust management

The TCG approach may be extended to manage trust relationships in the cloud through CoTs [1]. Central is the process of *remote attestation*: a *Trustor* (client) can decide based on provided evidence regarding configuration of a *Trustee* (server) on its authenticity and on its integrity, i.e., is the resource in a trustworthy state. Two main issues have to be resolved:

- *Interference between transparency and trust management:* due to multi-layering, if a CoT is broken, the user might need to become aware of infrastructure details which are normally hidden from him. This might take time to assess again the trustworthiness of the infrastructure. Thus, *trust anchors* needs to be established to be able to re-compute trust from intermediate points in layers, without starting again at the bottom in the physical layer.
- *Compositional CoTs:* when resources are managed as a group, e.g., as in domains, the issue is how to compute the group CoT from member CoTs.

It may be assumed that each abstraction (resources, domains, federated domains) of the infrastructure model has a RoT [1]. Attestation works by having each cloud resource endowed with a client agent (CA) monitoring the resource state and providing assurance of its trustworthiness and having a server agent (SA) located in the cloud manager to perform verification of resource trustworthiness.

- *Single resource CoTs:* In the physical layer, integrity is guaranteed using the TPM chip that can perform secure integrity measurement (e.g. reliable measurement, storage, and reporting). For the virtual and application layers, one option could be to rely on a virtual TPM [25]. However, the trustworthiness of the link between TPM and vTPM might be broken due to cloud dynamicity, e.g., VM creation and destruction. The proposed approach is to rely on the compositional CoT to derive the RoT of one layer from the CoTs of groups of resources in lower layers.

- *Compositional CoTs:* In the physical layer, for a single domain, the domain RoT is chosen as the cloud manager SA CoT. The domain CoT is essentially the combination of the CoT of that agent, and of the CoT of a resource in the domain. For multiple domains, the resulting CoT is the CoT of one of the member domains. In the virtual layer, the CoT of a domain is the combination of the CoT of the underlying federated domains in the physical layer, and of the CoTs of the resources member of the virtual layer domain. For the application layer, the approach is similar, this time relying on virtual layer domains instead of physical layer domains.

The overall result is a set of trust anchors at the application layer (for SaaS users), at the virtual layer (for PaaS users) and at the physical layer (for IaaS users) allowing to recompute trust from intermediate points, instead of from the ground up.
To implement the cross-layer trust management component, we are currently extending the VESPA self-protection framework [190] to implement such a trust management model for SUPERCLOUD to enable remote resource attestation.

7.4.3.3 Hardware security mechanisms

The *Trusted Platform Module* (TPM) is a dedicated chip which was designed according to the Trusted Computing Group (TCG) Specification. Figure 7.14 represents an approach utilizing a TPM to integrate trustworthy components in the hardware design.

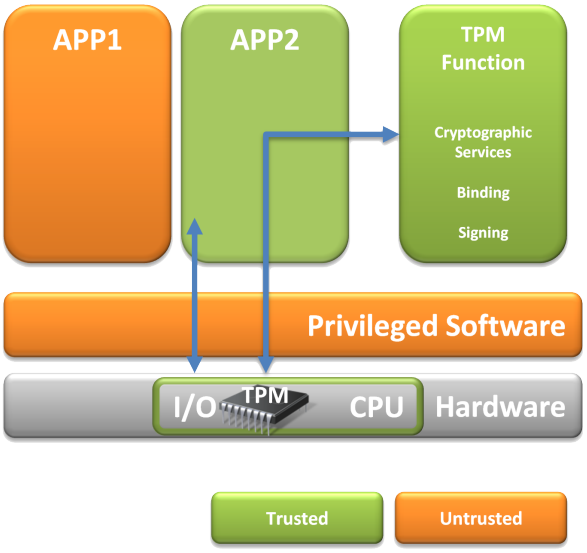


Figure 7.14: Usage of a TPM in the SUPERCLOUD architecture

The TPM *I/O block* is the interface which is connected by means of a so called *Low Pin Count bus* (LPC bus) with the mainboard and links the chip to the external world. A TPM offers technical features for the generation of asymmetric keys for RSA encryption and decryption for a key length up to 2048 bits. Each TPM can be exactly assigned to a unique *Endorsement Key* (EK) which approves the validity of a TPM. Related to the EK are the *Storage Root Key* (SRK) and *Attestation Identity Key* (AIK). The SRK protects application generated TPM keys to be unusable without the directed TPM. The AIK is only used for signing values which are stored in the volatile storage *Platform Configuration Register* (PCR). Further on, SHA-1 and MD-5 hash algorithms are supported by the chip and a physical random number generator is implemented too.
Not only the TPM, but even other architecture components need to be trustworthy to provide trust inside the module. The *Core Root of Trust Module* (CRTM) as part of the BIOS measures the integrity of the underlying code with a hash function and stores measured values secure and digitally signed in the TPM. For the communication between a TPM with an application, *binding* is used to encrypt data.

When binding is applied, the non-migratable SRK encrypted data is bound to a specific platform. To extend this function, the platform's state at the time of encryption [64] is added to the ciphertext and it can be decrypted when the platform has the state as it had when it was created. This method is called *sealing*.
In order to ensure that the *Chain of Trust* (CoT) is adhered and only trusted users may have access control, the TCG defines five types of credentials [2] to help to verify these permissions: Endorsement Credential, Validation Credential, Conformance Credential, Platform Credential and Attestation Identity Credential. The usage of a TPM and these credentials modifies a normal system to be a Trusted Platform (TP).
Although operations on critical data are performed inside the TPM and it protects against software attacks, a disadvantage is that a single TPM is connected to a single platform with a single owner which makes it ineffectual for cloud computing environments. A solution proposed in [25] illustrates a model where *virtual TPMs* (vTPM) are used on a layer over a trustworthy Trusted Virtual Machine Monitor (TVMM) hypervisor which binds securely the physical module to the virtual one. Each vTPM Manager which initializes a single or more vTPMs is built in a virtual machine (VM) that is located on the TVMM virtualization architecture. A vTPM can be only initialized, when the infrastructure with a physical TPM is considered as *trustworthy*. Otherwise, it cannot be installed. The advantage of this solution is the usage of only a single physical hardware module as a RoT to be used in cloud computing environment. Thereby the same features as with a physical TPM are usable, because a VM is bound as long as its lifetime to a virtual TPM instance [25].

7.5 Security self-management and orchestration

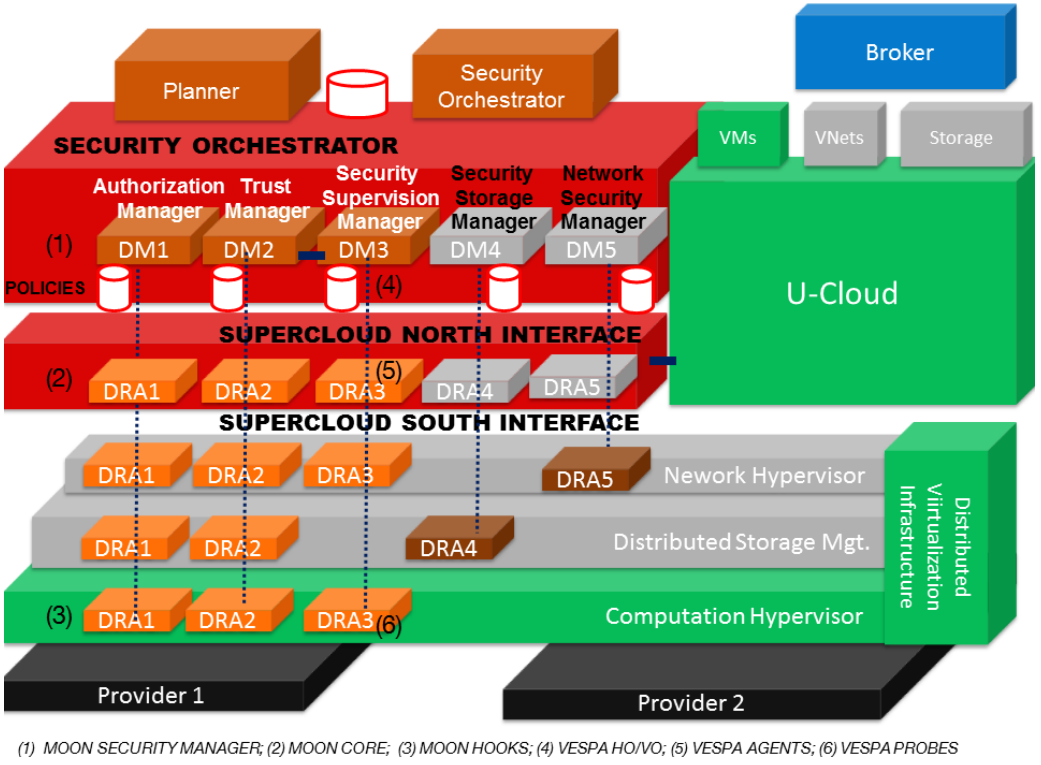


Figure 7.15: Overall self-management architecture for computation

Figure 7.15 gives a high-level overview of the components involved in self-management for computation, refining the overall WP1 self-management architecture. The architecture is layered, inspired from the NFV specification.

In this architecture, at the bottom is the *SUPERCLOUD hypervisor*, with computation, data, and network layers. The computation part has already been described previously. The hypervisor allows to run *U-Clouds* which contains computation, storage, or network resources, above multiple providers. We make the assumption that the virtualization layer contains *probes* (for detection), and *enforcement points* (for reaction), abstracted on the Figure as *Detection and Reaction Agents* (DRA). DRAs are considered security service-specific (e.g., for intrusion detection/prevention, authorization, trust management), and operating on a specific layer (computation, data, network). Some DRAs may be used for several security services. Some security services may also have DRAs in multiple computation/data/network layers.
Above is the *SUPERCLOUD Enforcement Point*. It provides a per-security service view of DRAs, aggregating detection/reaction independently from virtualization layers. It contains the overall security context of the infrastructure, or reaction plan to carry out, to be enforced in lower layers.
Finally, the *SUPERCLOUD Orchestrator* encapsulates security decision-making. It contains:

- Different *Security Services* such as authorization, trust management, or security supervision understood as intrusion detection/prevention for computation. Other security services such as a security storage (defined in WP3), or a network security (defined in WP4) may also be considered, but are outside the scope of WP2.
- An overall *Security Orchestrator* that defines how to compose together the different security services, in a similar way to the NFV Orchestrator.

A last component, the *Security Planner* is in charge of preparing the security response for the Security Orchestrator. It notably takes into account the security SLA requested by the user upon creation of the U-Cloud, and security constraints expressed by each underlying cloud provider.

7.5.1 User-centric security policy manager

The user-centric security management is carried out by an adaptation of MotOrBAC [63, 11]. MotOrBAC is a tool that allows users to specify and simulate their policies. The tool is build on the top of the OrBAC API which can be integrated to applications or cloud layers to interpret OrBAC policies. The general architecture of the MotOrBAC tool is depicted in the figure hereafter.
The OrBAC API provide several OrBAC policy implementations:

- The first implementation uses a custom inference engine which uses the join/fork framework from java 7 to provide an efficient derivation process. This is the default implementation suggested to the user when creating a policy. The policy is saved as a XML document where the abstract and concrete entities are stored.
- The second implementation is derived from the previous implementation and uses the same inference engine. The abstract policy is stored in an XML file and the concrete entities are stored in a Mysql database.

7.5.2 Planner

As highlighted above, the planner is responsible of extracting the user's description of the expected cloud configuration in terms of required components (and inter-connectivity), as well as the desired security and performance level. To that aim, the Planner will parse the active SLA to identify the configuration that needsto be deployed by the provider.

Parsing the SLA will allow the Planner to extract parameters that will be used to generate deployment templates. These templates are used to allocate, place and configure the instantiated user cloud.
The planner is also used by the Orchestrator to plan reactions to security threats such as attacks and intrusions,

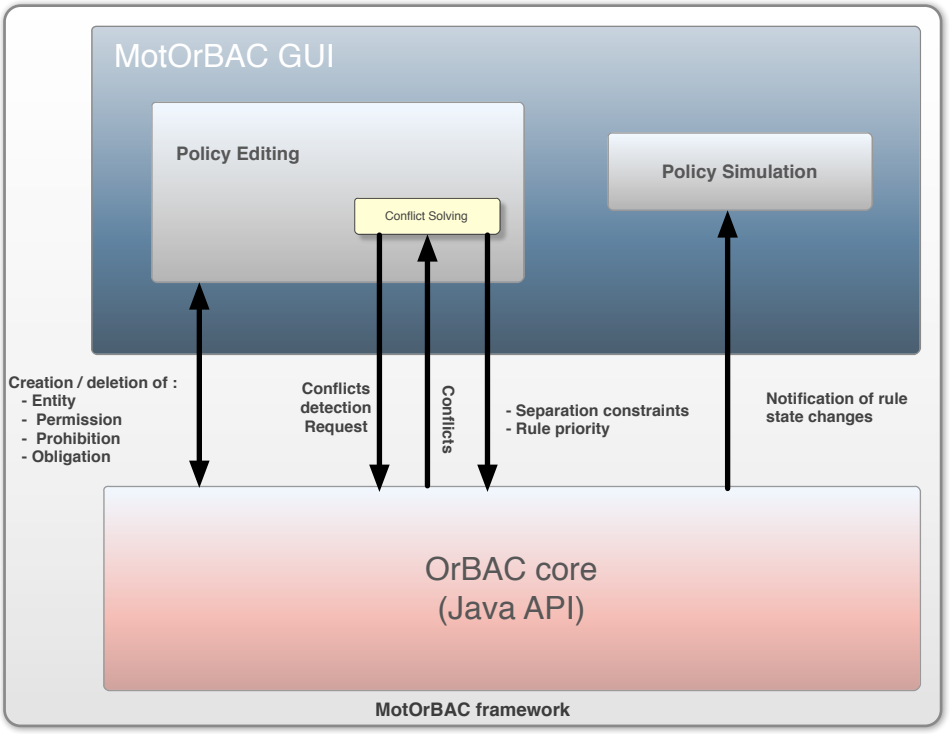


Figure 7.16: The MotOrBAC architecture

7.5.3 Security supervision framework

To implement vertical and horizontal SUPERCLOUD security self-management, we leverage on the VESPA (Virtual Environments Self-Protecting Architecture) framework [190]. VESPA explores self-protection to detect and react to threats in IaaS cloud infrastructures, with well-known benefits for security management automation such as lighter administration, lower incident response times, or reduced error-rates.

The VESPA framework

VESPA is built on the following principles: (1) *policy-based self-protection*, i.e., the architecture should be a refinement of a well-defined security adaptation model based on policies; (2) *cross-layer defense*, i.e., detection/reaction should not be performed in a single software layer, but may span several layers; (3) *multiple self-protection loops*, i.e., several control loops of variable granularity level should be defined and coordinated; and (4) *open architecture*, i.e., multiple detection and reaction strategies/mechanisms, e.g., off-the-shelf security components, should be easily integrated in the architecture, to mitigate both known and unknown threats.
VESPA regulates protection of resources through several coordinated autonomic security loops which monitor the different infrastructure layers. The result is a very flexible approach for self-protection. Its main features are: (1) policy-based security adaptation based on a self-protection model capturing symmetrically and flexibly both detection and reaction phases; (2) two-level tuning of security policies according to security contexts both inside a software layer and across layers; (3) flexible orchestration of layer-level self-protection loops using system-wide knowledge to allow a rich spectrum of overall infrastructure self-protection strategies; and (4) a layered, extensible architecture allowing simple integration of commodity detection and reaction components thanks to an agent-based mediation plane abstracting security component heterogeneity.
VESPA considers security management orthogonally to infrastructure layers. Self-protection is achieved through a set of autonomic loops operating over a number of components organized into four distinct planes, shown in (see Figure 7.17).

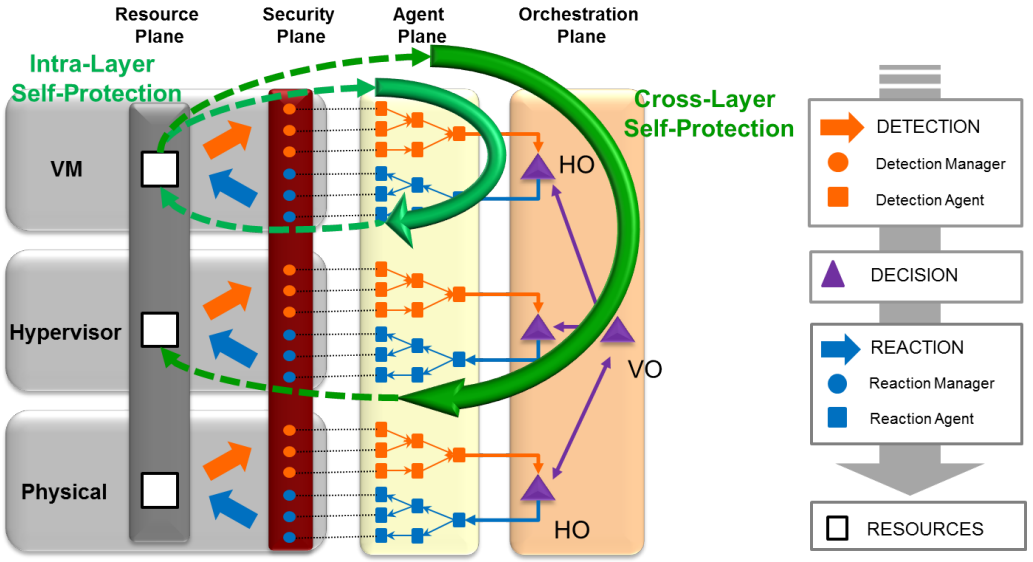


Figure 7.17: VESPA cross-layer self-protection

The *Resource Plane* contains the resources to be monitored and protected. The *Security plane* contains commodity detection and reaction components that deliver security services such as monitoring (e.g., IDS) or control (e.g., firewall) of resource behavior and/or state. The APIs of those components are usually vendor-specific. The *Agent Plane* abstracts away security component heterogeneity by defining a mediation layer between security services and decision-making elements. This plane is built from two hierarchies of agents, one for detection, and another for reaction. Leaf agents are adapters between the VESPA framework and the security components to translate vendor-specific APIs into a normalized format both for detection and reaction. They enable to plug-in third-party security components within the framework. Higher-level agents are in charge of alert correlation or of reaction policy refinement. The *Orchestration Plane* contains the decision-making logic. It is composed of two types of autonomic managers: *Layer Orchestrators (LOs)* perform layer-level security adaptation; *Vertical Orchestrators (VOs)* are in charge of cross-layer security management.

Extending VESPA for SUPERCLOUD security self-management

This architecture is particularly well suited for resource integrity monitoring and remediation. It is applicable both for cross-layer self-protection and for inter-cloud end-to-end protection [120]. In SUPERCLOUD, we plan to extend VESPA to also take into account: (1) more layers than a normal IaaS, due to the use of nested virtualization; (2) provider vs. user control arbitration; (3) cross-layer trust management. A sample VESPA-based architecture to implement cross-layer security self-management in the SUPERCLOUD virtualization architecture is shown in Figure 7.18.

7.6 Compliance manager

The most appealing traits of cloud computing are its scale and elasticity as users can request resources and obtain them quickly. In order to provide such service while remaining profitable, cloud operators rely on resource multiplexing over a large infrastructure. Naturally, this comes with a price: as operational complexity increases, mistakes, especially configuration errors, are bound to happen [148, 171, 80]. Downtimes caused by these mistakes are particularly costly [183, 6].
Additionally, the size of system and network administrator teams increases with the size of the infrastructure. In particular in virtualized infrastructures that are not yet fully automated, coordinating the administrative activities can be hard. Changes of different administrator may be dependent on

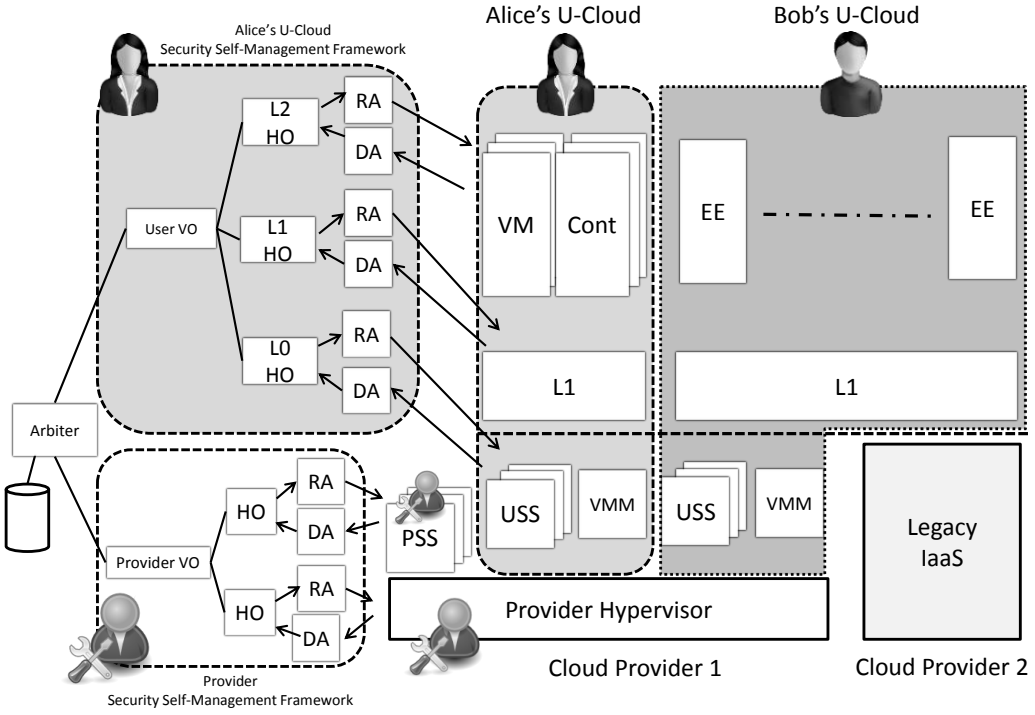


Figure 7.18: Integration of self-protection into virtualization architecture

each other or are even conflicting. Furthermore, administrators need to know who changed what in the infrastructure, particularly in cases of problems.
Even though concepts like automation and customer self-service may mitigate some management issues due to human intervention, the fundamental problems of misconfigurations and accountability are still present. Automation software itself could issue wrong changes to the infrastructure, either due to software bugs or a wrong configuration of the automation [146, 145, 53]. Assessing the underlying changes to the infrastructure with regard to security and function requirements is crucial for both fully automated as well as manually managed infrastructures.
Tool-support and systematic processes are required to tackle the problem of managing complex virtualized infrastructures. We observe that there is a natural correspondence between configuration management and software engineering; and that software engineering is a mature field, where large scale projects, holding several million lines of code, are actively being developed (Linux kernel, Eclipse [135], Mozilla [137], etc.). Drawing from the best practices in software engineering, we highlight a parallel to cloud configuration management. We show how processes, built around version control systems (VCSes) could be applied to cloud configuration management *at the infrastructure level.*
We propose a configuration management system, called CCTV, that cloud providers and their administrators could use to reduce misconfigurations, facilitate collaboration, and provides accountability. In particular, we aim for the following goals:

- *Change Reviews and Collaboration:* To prevent misconfigurations, changes are reviewed before applied to the infrastructure. Our tool allows both reviews done by human administrators as well as by an automated analysis of changes with regard to security policies or SLAs. In addition, multiple administrators could coordinate and be aware of each other changes. CCTV facilitates this collaboration and also detects conflicting changes.

- *What-If Analysis:* We allow the simulation of large infrastructure changes consisting of multiple steps for the current infrastructure. The resulting infrastructure state will be subject to automated analysis or human review before deployed to the infrastructure.

- *Accountability:* Our tool maintains who has changed what, when and how in the infrastructure. This supports failure diagnosis as well as the mitigation of insider threats.

These goals are in line with requirements for cloud compliance. We need to be able to: 1) specify a compliant state as a policy and have an automated way to check that; 2) perform change reviews (either manually or automated) to prevent that the system becomes incompliant; and 3) provide accountability of infrastructure changes to document who changed what. Our proposal offers an automated approach in supporting operators of virtualized and cloud infrastructures in their compliance fulfillment.

7.6.1 CCTV - the core concepts

CCTV (Cloud Configuration Through Versioning) is based on three core concepts that establish the link between best practices in software engineering and cloud configuration management through the use of version control systems.

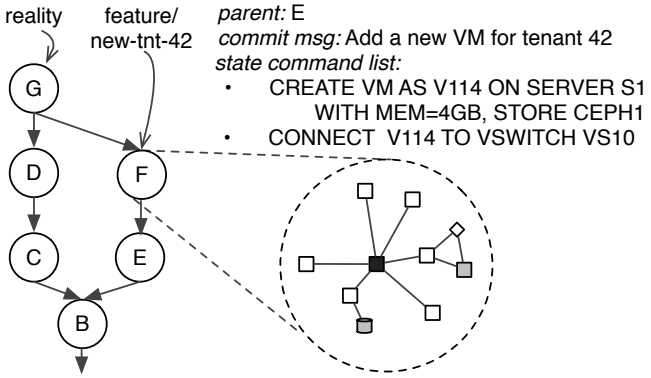


Figure 7.19: Example of two branches, 'reality' and 'feature/new-tnt-42'. For state F, we illustrate the graph stored inside, as well as other content of state F in the repository.

7.6.1.1 A model of the virtualized infrastructure as state

In software development, a state in a VCS contains the source code in the form of files and directories. In our case, the state is the topology and configuration of a virtualized infrastructure in the form of a graph model. The typed nodes of such a graph are infrastructure elements, e.g., virtual machines, virtual switches, but also physical hosts (hypervisors). The edges represent topological relationships between nodes, e.g., the fact of a virtual machine running on a hypervisor is represented by an edge. The graph model is populated in an automated way by extracting the configuration of hypervisors or management hosts of virtualized infrastructures, and then translating the configuration into graph nodes and edges [38]. The configuration of virtual machines is not a part of this model and hence CCTV is complementary to existing configuration management tools such as Puppet [155], Chef [55], CFengine [52].

7.6.1.2 Maintaining a history of infrastructure states

A developer in a software project changes the source code by editing files and performs a commit to the version control system describing the changes, which leads to a new state of the code in the repository. CCTV follows a similar paradigm where an agent, such as an administrator or an automation software, changes the configuration of the infrastructure through a cloud management API, that provides a set of operations to transform the infrastructure. Our tool provides an interface to obtain such changes in a form of a set of operations with their arguments, that is, a change plan. These sets of operations are then applied to the infrastructure and a new state is created. Furthermore, CCTV provides a mechanism to monitor the infrastructure for changes and correlate them to operations.

Maintaining such a history of the infrastructure evolution provides enhanced accountability (e.g., determining who was responsible for certain changes), as well as analysis opportunities in terms of metrics (e.g., did the utilization or security compliance degrade over time). Furthermore, "good" states of the infrastructure can be explicitly tagged and certain erroneous changes reverted back to a good state, assuming deterministic operations.

7.6.2 What-if scenarios based on branching

The concept of branches in VCS is often used to develop new features in separate branches and merging them back into a main branch once a feature is completed and properly tested. In CCTV, we follow a similar branching concept where a feature is, for instance, the creation of a new virtual tenant infrastructure (creating VMs, creating virtual networks etc.). In order to determine how the infrastructure evolves in a feature branch, we model operations of the cloud management API in terms of how they change the graph model of the infrastructure. A branch can be analyzed with regard to SLAs, security policies, functional requirements etc, before considered completed. Then, merging the branch back into the main branch, which we call *reality*, actually deploys all the planned changes of the feature branch.
Existing work on what-if scenario analysis focuses on network aspects [182, 5]. CCTV targets virtual infrastructures and considers the entire stack (compute, network, storage). Early results and approaches on the modeling of operations and their analysis have been proposed by Bleikertz et al. [36].

7.6.3 Deployment environments

We target two environments of virtualized infrastructures where CCTV can be deployed: Non- or semi-automated infrastructures as well as fully automated ones. In both environments, *agents*, i.e., human administrators or automation software, perform changes on the virtualized infrastructure. These changes can be problematic either due to misconfigurations by an administrator or due to misbehavior by the automation software. CCTV provides for both environments accountability of changes as well as the analysis of a change on the infrastructure with regard to security, SLAs, or other requirements.

7.6.3.1 Non/semi-automated virtualized infrastructures

In this environment we are dealing with medium-sized infrastructures that are typically found in enterprise environments, which leverage virtualization for higher utilization and efficient management. These environments face no automation or only partial automation, and human administrators are responsible for managing the environment. The particular challenges in this environment are misconfigurations by administrators due to the complexity of the environment, accountability and history of changes, as well as collaborative administration.

7.6.3.2 Fully automated infrastructure clouds

Infrastructure clouds are fully automated and large-scale virtualized infrastructures. Automation software, such as OpenStack [147], perform resource provisioning and apply the required changes to the infrastructure, such as spawning new virtual machines or re-configuring the virtual networking components. Typically, the automation software uses the same interfaces for configuring the virtualized infrastructure as a human administrator would do through a client, which means that both environments behave similar for the underlying infrastructure changes. In this environment infrastructure misconfigurations may happen due to bugs in the automation software, since they are complex itself, or that the software was wrongly configured [146, 145, 53]. The goal of CCTV is to prevent such misconfigurations and to provide insights on how the infrastructure evolved over time in terms of security and other requirements.

7.7 Authorization

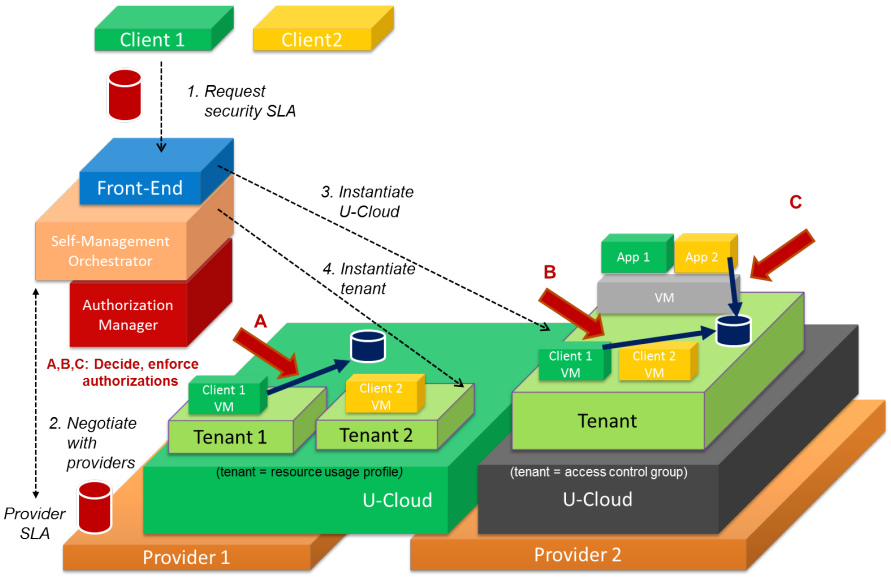


Figure 7.20: Different levels of authorization

7.7.1 Design of the authorization component

Different complementary visions and levels of authorization may be considered as shown in Figure 7.20. Central is the notion of *tenant*, which may range from a usage profile over resources, to an access control group including several users, or to simply a user. Authorization may be:

- *Usage control oriented (A):* the user requests a security SLA to the SUPERCLOUD which will instantiate the U-Cloud accordingly after negotiating with underlying providers. A tenant is then instantiated, giving usage rights to resources to VMs running within the tenant. This is addressed by the OrBAC authorization framework (Section 7.7.2)[3].
- *Access control oriented (B):* the authorization component controls access to different VMs running in the tenant according to an access control policy. This is addressed by the MOON authorization framework (Section 7.7.3).
- *Application-level (C):* the authorization component controls access to resources between different applications. This is addressed by our third authorization framework (Section 7.7.4).

7.7.2 Usage control oriented authorization: OrBAC framework

The user-centric of security control advocated in the SUPERCLOUD project calls for a an expressive and flexible *authorization* and *access control* framework. To that aim, we propose to adapt the **Organization Based Access Control** (OrBAC) infrastructure to model and specify policies. As presented in Chapter 5, OrBAC aims at modeling security policies that are centered on organizations. Intuitively, an organization is any entity responsible for defining and/or managing a security policy. Hence a user cloud can be considered as an organization, but security components such as firewalls or virtual machines managers can also be modeled as organizations as well.

[3]This framework may also address access control-oriented authorization (vision B).

The OrBAC framwork architecture is composed of three elements; (i) the OrBAC API, (ii) the *MotorBAC Tool* and (iii) the OrBAC Plugins, as show in Figure 7.21 below. This is mainly a conceptual view of the OrBAC infrastructure, as the three components do not necessary require to be on one single point within the SUPERCLOUD self-management architecture.
In the following we give a description of the *OrBAC API* [107] and the *OrBAC Plug-ins* [63, 11]. We refer the reader to the Section 7.5.1 for a presentation of the *MotorBAC* policy Tool.

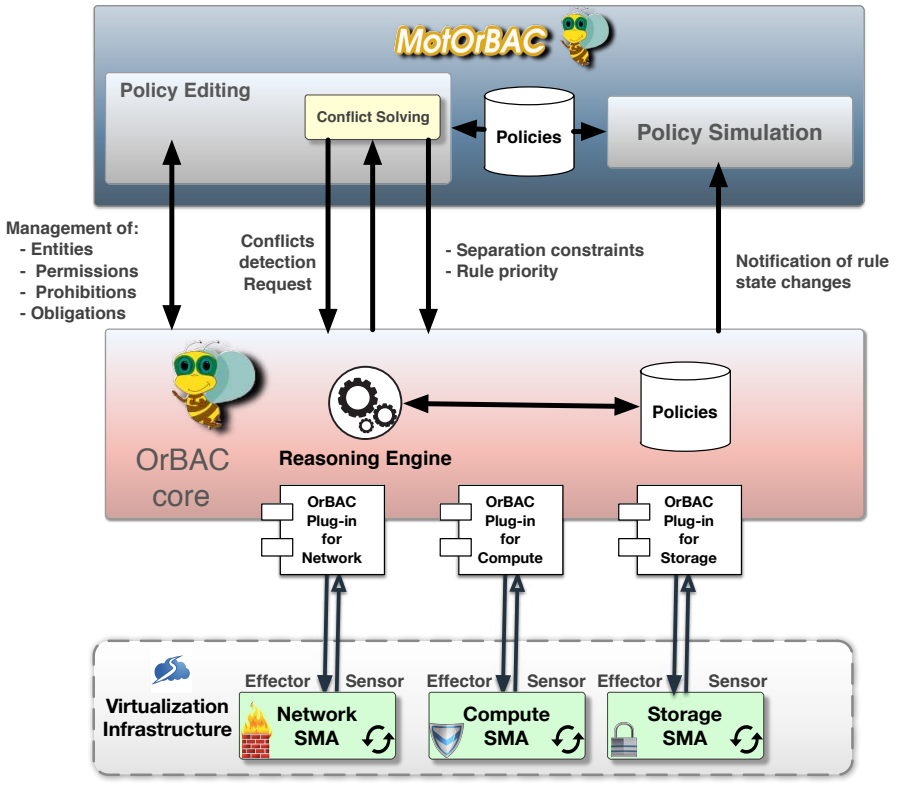


Figure 7.21: Architecture of the OrBAC authorization framework

OrBAC API

OrBAC Application Programmer Interface (OrBAC API) [107] is the core component of the OrBAC framework. It provides also a Java library that can be used to create and manipulate OrBAC policies within Java Applications. The API features the following OrBAC policy editing capabilities:

- Abstract policy specification: organizations, roles, activities, views, contexts, and abstract rules (permissions, obligations, prohibitions) can be manipulated. This includes organizations, roles, activities, and views hierarchies
- Separation constraints and rules priorities can be specified to solve conflicts between abstract rules
- Several languages can be used to express contexts and entity definitions. Simple ad-hoc languages have been defined to express temporal conditions or simple conditions on concrete entities (subject, action or object) attributes. Two more powerful languages can be used, Java and Prolog, to be able to express a wide range of conditions
- The administration policy, or AdOrBAC policy, associated to an OrBAC policy can be specified using the same concepts and API methods

OrBAC plug-ins

Neither the MotOrBAC nor the OrBAC API do include any functionality to deploy and/or enforce OrBAC policies. To that aim, the pluggins have been proposed to extend the OrBAC infrastructure with deployment mechanisms. Currently a short list of publicly available OrBAC plug-ins exists but this list should be extended with SUPERCLOUD specific plug-ins.

7.7.3 Access control oriented authorization: MOON framework

SUPERCLOUD allows building dynamic architectures and provisioning different cloud resources/services for cloud service consumers. Trust over the provided cloud architecture/service/resource thus becomes a new challenge. To avoid losing control over IT assets that consumers put in the cloud, a dedicated security management sub-system called MOON has been developed. The administrator can define high-level security policies, which can be enforced across the multi-cloud infrastructure.

A policy-based architecture

MOON can be considered as a management layer over OpenStack. It enables to dynamically create security modules and assign these modules to protect different tenants in OpenStack. The MOON high-level design and architecture are shown in Figure 7.22. The core part of MOON is its *policy engine*. The policy engine is generic to support a large set of security models used by customers, and robust so that all manipulations on the policy engine need to be proved correct. An extensible attribute-based access control model called EMTAC was designed to formalize policy specification, policiesdministration, inter-policy collaboration (, anits corresponding administration policies).

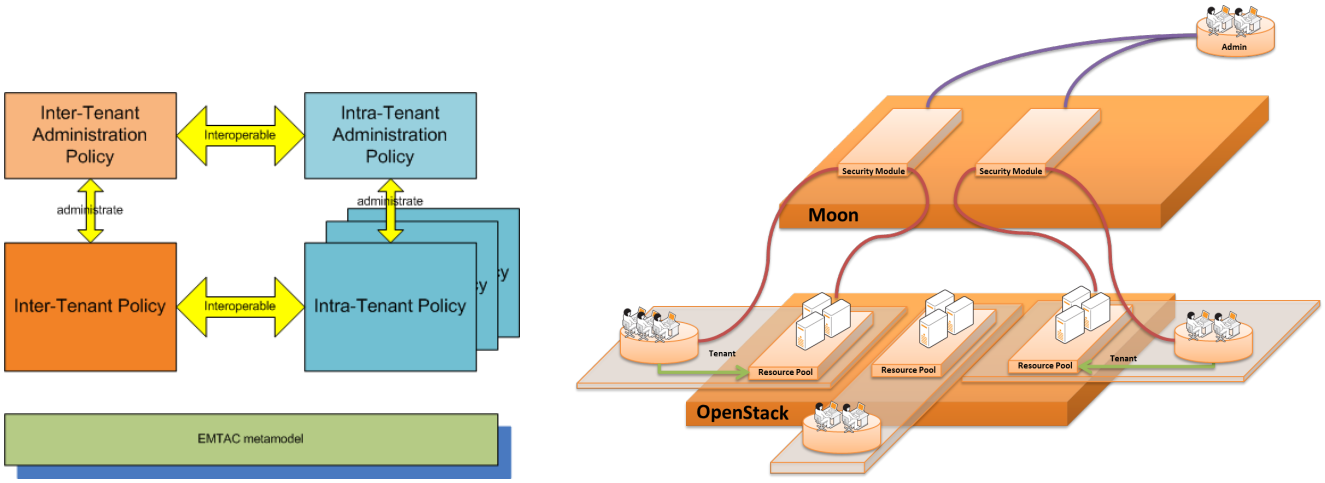


Figure 7.22: MOON:(a) principle; (b) architecture

Implementation

MOON (second release) proposes a dynamic solution to protect each tenant focusing on authorization that is enforced thanks to installation of a dedicated module in the management plane. All user operations are then controlled and validated based on this security module. Several additional protection services are foreseen in future releases. For instance:

- *Logging* of operations of users and interactions between VMs for traceability and accountability.
- *Intrusion detection* and run-time security monitoring.
- *Network-level security enforcement* to enable network protection for SDN controllers. Based on a defined policy, this feature may allow to configure Neutron, OpenDayLight, etc.

- *Storage-level security enforcement* to controlling access to storage blocks or files. Stored data may also be encrypted based on MOON security policies before transmission over a risky network.

7.7.4 Application-level authorization framework

In this section, we describe our authentication and authorization platform for cloud applications. See Figure 7.23 for an overview of the concepts discussed below. The authorization model consists of three main concepts: the *identity* of the user of a cloud application; the *application* that the user is running; and any *APIs* (e.g., storage or computation) offered by the cloud platform that the application relies on. At a high level, access to API calls itself is based on authentication both of the user identity and of the application. If access to an API call for a particular identity using a particular application is granted (e.g., to the storage API), then along with the API call, information about the identity and the application is supplied, which the API can then use to determine whether the particular request from the API call (e.g., "get `patientrecords/1047294/scans/4333`") should be satisfied. This second question, i.e., whether the particular request should be satisfied, is considered application-dependent and hence not part of the authorization framework. We now describe the concepts determining access to APIs in more detail.

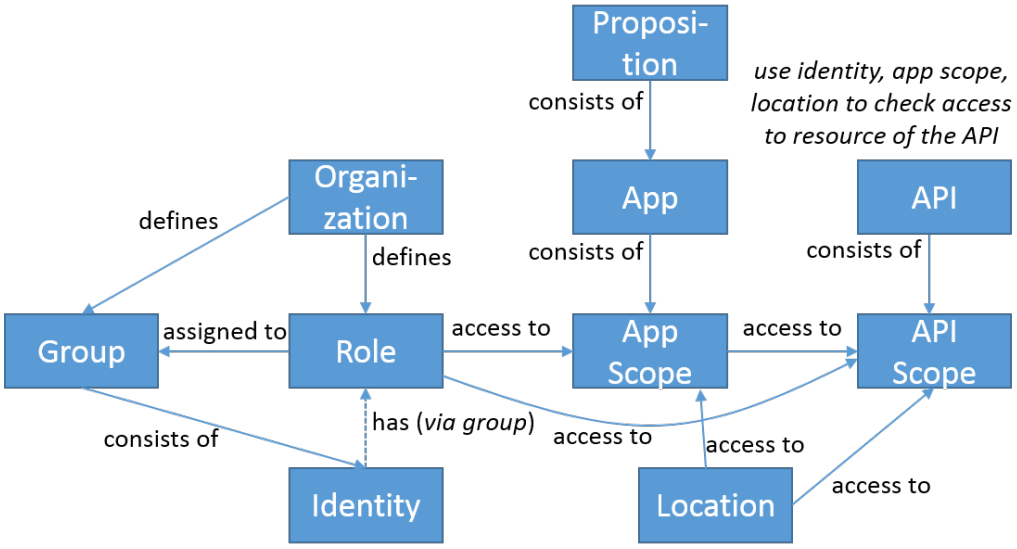


Figure 7.23: Application-level authorization framework: concepts and relations

APIs

An API is any service offered by a cloud application platform, such as storage or computation. Each API consists of a number of *API scopes*. A scope is a logical group of API calls at which access rights for applications are arranged. For instance, a storage functionality may have UPLOAD, DOWNLOAD, and CONFIG scopes.

Applications

Applications are pieces of functionality that are used to access APIs. For instance, applications can be web applications, mobile applications, or desktop applications. Together, several application are part of a *proposition*, i.e., a product that provides a specific offering to the market. Applications are divided into *application scopes*: logical sets of functionality that require certain access permissions from an API. Applications (more specifically, specific *versions* of an application) authenticate to an API in order to gain access using an application specific secret.

Identities

Identities are organized through *organizations* (e.g., in the medical domain: a particular hospital). An organization defines several *groups* (e.g., doctors or nurses) of which identities are members. The organization also defines several *roles* (e.g., administrator, users) which are assigned to these groups. Hence, a user has the roles that have been assigned to the group(s) that he is a member of.

Authorization

Suppose an applications wants to obtain access to an API call on behalf of a certain user. Authorization happens both at application level (does the user have access to a certain application?), and at API level (does the application representing the user have access to a certain API call?). Having authorization at both places is particularly important when an API exposes (services based on) the same data to several applications: in this case, one application should not have to depend on the authorization mechanisms of other applications, hence authorization should be performed by the API (and, apart from that, the application should of course be able to decide to which users to offer its services).
Authorization at the application level depends in the identity of the user, and the location where the user resides. As discussed in Deliverable D1.2, in practice, the application redirects the user to an IAM (identity and access management) server to obtain authorization to act on behalf of the user for a given application scope. The IAM authenticates the user and, if needed, obtains explicit authorization (consent) from the user. Based on that, a policy-based decision is made whether the user has access to the application, e.g., using role-based access control (RBAC), a check is performed whether the user has the role needed to obtain access. As indicated in Section 2 (e.g., design requirements DR8, DR9), apart from the identity/role of the user, also his physical location is important for authorization decisions. Therefore, also the physical location of the user is taken into account.
For the purpose of this model, API-level authorization aims to answer the question whether the application representing the user has access to certain API calls. This allows the API to only expose services to known and trusted applications, a necessity when it wants to restrict the purpose for which sensitive information is exposed. In practice, the application authenticates to the API using the application secret. Moreover, the API obtains an "access token" containing claims about the application (scope, version, etc.) itself and user (identifier, organization, location, etc.). This information allows the actual check to take place that, e.g., a user is entitled to access a particular file, is performed. This final decision is API-dependent and considered out of scope of the application-level authorization framework, but may in practice be also be policy-based, e.g., using RBAC.

Chapter 8 Conclusions

Summary

This deliverable presented a preliminary architecture for a virtualization infrastructure enabling secure computation with self-managed protection across heterogeneous cloud providers.

- We first analyzed the design requirements of the architecture. Three main protection challenges stand out: (1) *security* in layers of the infrastructure, with high level of *control* by customers; (2) *interoperability* for distributed secure computation across clouds; and (3) overcoming *security administration complexity* through unified security automation. Those challenges may alternatively be formulated requiring the infrastructure to provide *self-service*, *self-managed*, and *end-to-end security*, with also *resilience* guarantees.
- We then surveyed the state of the art of cloud security and virtualization key enabling technologies to build this architecture, notably: virtualization and isolation for building the computation hypervisor and its services, and access control models and policies, trust management, and self-management of security to build the self-protection infrastructure for VMs.
- Finally, we presented a preliminary architecture for the virtualization and self-management infrastructure, describing the overall design, the architecture of the different components, also showing how the different techniques enable to fulfill the identified requirements.

Assessment

- For *self-service security*, the SUPERCLOUD computation hypervisor provides a distributed resource abstraction layer to federate computations across providers. Its layered and modular design enables to meet interoperability and security requirements, separating provider and user control over VM security between the SUPERCLOUD and distributed clouds. The self-management infrastructure also enables distributed and automated control of security of compute resources, notably thanks to multi-provider policies for customizing security of user-centric clouds, control being tunable between user and provider.
- For *self-managed security*, the SUPERCLOUD self-management infrastructure provides automated cloud security monitoring, vertically across layers, to guarantee VM security even if some layers are untrusted, and horizontally across cloud provider domains, to guarantee interoperability of security mechanisms such as authorization.
- For *end-to-end security*, the architecture provides unified security and trust for the distributed cloud. The SUPERCLOUD virtualization architecture relies on different isolation technologies, software and hardware, to guarantee distributed secure computation over untrusted environments, upon which end-to-end security SLAs can be preserved thanks to the self-management infrastructure. A trust management framework also guarantees authenticity and integrity of the link between a VM and its cloud resources, across layers, provider domains, and SUPERCLOUD users, composing chains of trust, and relying on hardware mechanisms.

- Finally, in terms of *resilience*, the architecture provide some guarantees of distributed virtualized infrastructure state correctness, through an automated configuration compliance mechanism, highly relevant for prevention of violations and accountability.

Next steps

Next steps will be to implement and evaluate the proposed computation architecture and its components. This notably includes refining the relation with the use cases, and the other architectures for data protection (WP3), networking (WP4), and overall design (WP1). The overall trust model for the computation infrastructure will also be refined. During this phase, the platform architecture will be continuously re-evaluated and improved, e.g., refining the description of some components of this preliminary version of the computation architecture.

List of Abbreviations

ABAC	Attribute-Based Access Control
ACM	Access Control Matrix
AIK	Attestation Identity Key
API	Application Programming Interface
ATN	Automated Trust Negotiation
BGP	Border Gateway Protocol
BIOS	Basic Input Output System
BM	Bare-metal
CaaS	Client-Controlled Cryptography-as-a-Service
CBH	Component-Based Hypervisor
CCTV	Cloud Configuration Through Versioning
CFI	Control-Flow Integrity
CMDB	Configuration Management Database
CoT	Chain of Trust
CPU	Central Processing Unit
CRTM	Core Root of Trust Module
CTL	Computation Tree Logic
DA	Detection Agent
DCA	Distributed Cloud Architecture
DCC	Distributed Cloud Computing
DCI	Distributed Cloud Infrastructure
DIFC	Decentralized Information Flow Control
DM	Decision Manager
DoS	Denial of Service
DR	Design Requirement
DRA	Detection and Reaction Agent
DTM	Decentralized Trust Management
EC	European Commission
EC2	(Amazon) Elastic Compute Cloud
EE	Execution Environment

EK	Endorsement Key
ER	Emergency Room
FSDT	Fuzzing Semantic Decision Tree
GPDR	General Data Protection Regulation
GPH	General-Purpose Hypervisor
HO	Horizontal Orchestrator
HVM	Hardware Virtual(ized) Machine
IaaS	Infrastructure-as-a-Service
IBAC	Identity-Based Access Control
ICU	Intensive Care Unit
IDS	Intrusion Detection System
IFC	Information Flow Control
IOMMU	Input/Ouput Management Memory Unit
IPC	Inter-Process Communication
IVP	Integrity Verification Proxy
IT	Information Technology
LL	Lower Layer
LO	Layer Orchestrator
LPC	Low Pin Count
MAC	Mandatory Access Control
MH	Micro-Hypervisor
MMU	Memory Management Unit
NaaS	Network-as-a-Service
NFV	Network Function Virtualization
NFVI	NFV Infrastructure
NuSMV	New Symbolic Model Verifier
NV	Nested Virtualization
OrBAC	Organization-Based Access Control
OS	Operating System
PC	Personal Computer

PCR	Platform Configuration Register
PoP	Point-of-Presence
PSS	Provider System Services
PV	Para-virtualization
QoS	Quality of Service
RA	Reaction Agent
RBAC	Role-Based Access Control
RoT	Root of Trust
SDN	Software-Defined Networking
SGX	(Intel) Software Guard Extension
SLA	Service-Level Agreement
SMV	Symbolic Model Verifier
SRK	Storage Root Key
SSC	Self-Service Cloud
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TEE	Trusted Execution Environment
TP	Trusted Platform
TPM	Trusted Platform Module
TVD	Trusted Virtual Domain
TVDc	Trusted Virtual Datacenter
TVMM	Trusted Virtual Machine Monitor
TXT	(Intel) Trusted Execution Technology
U-Cloud	User-Centric Cloud
UL	Upper Layer
UML	User Mode Linux
USS	User-Centric System Services
VCS	Version Control System
VESPA	Virtual Environments Self-Protecting Architecture
VM	Virtual Machine

VMI	Virtual Machine Introspection
VMM	Virtual Machine Monitor
VMX	Virtual-Machine Extension
VNF	Virtualized Network Function
VO	Vertical Orchestrator
vTPM	virtual TPM
XML	Extensible Markup Language

Bibliography

[1] I. Abaddi. Clouds Trust Anchors. In *IEEE International Conference on Trust, Security, and Privacy in Computing and Communications (TrustCom)*, 2012.

[2] Mohammed Achemlal, Said Gharout, and Chrystel Gaber. Trusted Platform Module as an Enabler for Security in Cloud Computing. In *Conference on Network and Information Systems Security (SAR-SSI)*, pages 1–6, 2011.

[3] S. Al-Haj and E. Al-Shaer. A Formal Approach for Virtual Machine Migration Planning. In *Network and Service Management (CNSM), 2013 9th International Conference on*, pages 51–58, Oct 2013.

[4] J.M. Alcaraz Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray. Toward a multi-tenancy authorization system for cloud services. *Security Privacy, IEEE*, 8(6):48–55, Nov 2010.

[5] Richard Alimi, Ye Wang, and Y. Richard Yang. Shadow configuration as a network management primitive. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*. ACM, 2008.

[6] Amazon AWS. Summary of the Amazon EC2 and Amazon RDS service disruption in the US East Region. `http://aws.amazon.com/message/65648/`, 2011.

[7] Amazon AWS ECB. `http://aws.amazon.com`.

[8] Amazon Web Services. AWS Config. Available at `https://aws.amazon.com/config/`, 2014.

[9] G. Ammons, Jonathan Appavoo, Maria Butrico, Dilma Da Silva, David Grove, Kiyokuni Kawachiya, Orran Krieger, Bryan Rosenburg, Eric Van Hensbergen, and Robert W. Wisniewski. Libra: A Library Operating System for a JVM in a Virtualized Execution Environment. In *VEE'07*.

[10] A. Arefin and Guofei Jiang. CloudInsight: Shedding Light on the Cloud. In *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on*, pages 219–228, Oct 2011.

[11] Fabien Autrel, Frederic Cuppens, Nora Cuppens-Boulahia, and Céline Coma. MotOrBAC 2: a security policy tool. In *Third Joint Conference on Security in Networks Architectures and Security of Information Systems (SARSSI)*, 2008.

[12] AVISPA. The Intermediate Format. Deliverable D2.3, Automated Validation of Internet Security Protocols and Applications (AVISPA), 2003. `http://www.avispa-project.org/delivs/2.3/d2-3.pdf`.

[13] Ahmed M. Azab, Peng Ning, Emre C. Sezer, and Xiaolan Zhang. HIMA: A Hypervisor-Based Integrity Measurement Agent. In *Proceedings of the 2009 Annual Computer Security Applications Conference*, ACSAC '09, pages 461–470, Washington, DC, USA, 2009. IEEE Computer Society.

[14] Ahmed M. Azab, Peng Ning, Zhi Wang, Xuxian Jiang, Xiaolan Zhang, and Nathan C. Skalsky. HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 38–49, New York, NY, USA, 2010. ACM.

[15] Yossi Azar, Seny Kamara, Ishai Menache, Mariana Raykova, and Bruce Shepard. Co-location-resistant clouds. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, CCSW '14, pages 9–20, New York, NY, USA, 2014. ACM.

[16] J. Bacon, D. Eyers, T.F.J.-M. Pasquier, J. Singh, I. Papagiannis, and P. Pietzuch. Information flow control for secure cloud computing. *Network and Service Management, IEEE Transactions on*, 11(1):76–89, March 2014.

[17] Mirza Basim Baig, Connor Fitzsimons, Suryanarayanan Balasubramanian, Radu Sion, and Donald E. Porter. Cloudflow: Cloud-wide policy enforcement using fast vm introspection. In *Proceedings of the 2014 IEEE International Conference on Cloud Engineering*, IC2E '14, pages 159–164, Washington, DC, USA, 2014. IEEE Computer Society.

[18] P. Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *SOSP'03*.

[19] Yair Bartal, Alain Mayer, Kobbi Nissim, and Avishai Wool. Firmato: A novel firewall management toolkit. *ACM Trans. Comput. Syst.*, 22(4):381–420, November 2004.

[20] Moritz Y. Becker. Cassandra: flexible trust management and its application to electronic health records. Technical Report UCAM-CL-TR-648, University of Cambridge, 2005.

[21] Abdeltouab Belbekkouche, Md. Mahmud Hasan, and Ahmed Karmouch. Resource discovery and allocation in network virtualization. *Communications Surveys Tutorials, IEEE*, 14(4):1114–1128, Fourth 2012.

[22] D.E. Bell and L.J. LaPadula. Secure Computer Systems: Mathematical Foundations and Model, 1975. Mitre Corp. Report No. M74-244.

[23] J. Bellessa, E. Kroske, R. Farivar, M. Montanari, K. Larson, and R.H. Campbell. NetODESSA: Dynamic Policy Enforcement in Cloud Networks. In *Reliable Distributed Systems Workshops (SRDSW), 2011 30th IEEE Symposium on*, pages 57–61, Oct 2011.

[24] Muli Ben-Yehuda, Michael D Day, Zvi Dubitzky, Michael Factor, Nadav Har'El, Abel Gordon, Anthony Liguori, Orit Wasserman, and Ben-Ami Yassour. The turtles project: Design and implementation of nested virtualization. In *OSDI*, volume 10, pages 423–436, 2010.

[25] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vTPM: Virtualizing the Trusted Platform Module. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[26] Stefan Berger, Ramón Cáceres, Dimitrios Pendarakis, Reiner Sailer, Enriquillo Valdez, Ronald Perez, Wayne Schildhauer, and Deepa Srinivasan. TVDc: Managing Security in the Trusted Virtual Datacenter. *SIGOPS Oper. Syst. Rev.*, 42(1):40–47, 2008.

[27] E. Bertino, E. Ferrari, and A. Squicciarini. X -tnl: An xml-based language for trust negotiations. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 81–, Washington, DC, USA, 2003. IEEE Computer Society.

[28] E. Bertino, E. Ferrari, and A. C. Squicciarini. Trust-x: A peer-to-peer framework for trust establishment. *IEEE Trans. on Knowl. and Data Eng.*, 16(7):827–842, July 2004.

[29] KhalidZaman Bijon, Ram Krishnan, and Ravi Sandhu. A formal model for isolation management in cloud infrastructure-as-a-service. In ManHo Au, Barbara Carminati, and C.-C.Jay Kuo, editors, *Network and System Security*, volume 8792 of *Lecture Notes in Computer Science*, pages 41–53. Springer International Publishing, 2014.

[30] T. Binz, U. Breitenbucher, O. Kopp, and F. Leymann. Automated discovery and maintenance of enterprise topology graphs. In *Service-Oriented Computing and Applications (SOCA), 2013 IEEE 6th International Conference on*, pages 126–134, Dec 2013.

[31] Tobias Binz, Christoph Fehling, Frank Leymann, Alexander Nowak, and David Schumm. Formalizing the cloud through enterprise topology graphs. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, pages 742–749, Washington, DC, USA, 2012. IEEE Computer Society.

[32] Matt Blaze, Joan Feigenbaum, and AngelosD. Keromytis. Keynote: Trust management for public-key infrastructures. In Bruce Christianson, Bruno Crispo, WilliamS. Harbison, and Michael Roe, editors, *Security Protocols*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–63. Springer Berlin Heidelberg, 1999.

[33] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, SP '96, pages 164–, Washington, DC, USA, 1996. IEEE Computer Society.

[34] Sören Bleikertz, Sven Bugiel, Hugo Ideler, Stefan Nürnberger, and Ahmad-Reza Sadeghi. Client-controlled cryptography-as-a-service in the cloud. In *Applied Cryptography and Network Security*, pages 19–36. Springer, 2013.

[35] Sören Bleikertz and Thomas Groß. A Virtualization Assurance Language for Isolation and Deployment. In *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY'11)*. IEEE, Jun 2011.

[36] Sören Bleikertz, Thomas Groß, and Sebastian Mödersheim. Modeling and Analysis of Dynamic Infrastructure Clouds. Technical Report RZ3859, IBM Research, 2013.

[37] Sören Bleikertz, Thomas Groß, Sebastian Mödersheim, and Carsten Vogel. Proactive Security Analysis of Changes in Virtualized Infrastructures. In *Annual Computer Security Applications Conference (ACSAC 2015)*. ACM, Dec 2015.

[38] Sören Bleikertz, Thomas Groß, Matthias Schunter, and Konrad Eriksson. Automated Information Flow Analysis of Virtualized Infrastructures. In *16th European Symposium on Research in Computer Security (ESORICS'11)*. Springer, Sep 2011.

[39] Sören Bleikertz, Thomas Groß, and Carsten Vogel. Cloud Radar: Near Real-Time Detection of Security Failures in Dynamic Virtualized Infrastructures. In *Annual Computer Security Applications Conference (ACSAC 2014)*. ACM, Dec 2014.

[40] Sören Bleikertz, Matthias Schunter, Christian W. Probst, Konrad Eriksson, and Dimitrios Pendarakis. Security Audits of Multi-tier Virtual Infrastructures in Public Infrastructure Clouds. In *ACM Cloud Computing Security Workshop (CCSW'10)*. ACM, Oct 2010.

[41] Guido Boella and Leendert van der Torre. Role-based rights in artificial social systems. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, IAT '05, pages 516–519, Washington, DC, USA, 2005. IEEE Computer Society.

[42] Piero Bonatti, J. L. De Coi, Daniel Olmedilla, and Luigi Sauro. A rule-based trust negotiation system. *IEEE Trans. on Knowl. and Data Eng.*, 22(11):1507–1520, November 2010.

[43] Piero Bonatti and Pierangela Samarati. Regulating service access and information release on the web. In *Proceedings of the 7th ACM conference on Computer and communications security*, CCS '00, pages 134–143, New York, NY, USA, 2000. ACM.

[44] Piero A. Bonatti and Pierangela Samarati. Regulating Service Access and Information Release on the Web. *Journal of Computer Security*, 10(3):241 – 271, 2002.

[45] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. In *First Workshop on Mobile Cloud Computing (MCC)*, 2012.

[46] Christopher Burnett. *Trust Assessment and Decision-Making in Dynamic Multi-Agent Systems*. PhD thesis, University of Aberdeen, 2011.

[47] Shakeel Butt, H. Andrés Lagar-Cavilla, Abhinav Srivastava, and Vinod Ganapathy. Self-service Cloud Computing. In *ACM Conference on Computer and Communications Security*, CCS'12, pages 253–264, New York, NY, USA, 2012. ACM.

[48] Serdar Cabuk, Chris I. Dalton, HariGovind Ramasamy, and Matthias Schunter. Towards automated provisioning of secure virtualized networks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 235–245, New York, NY, USA, 2007. ACM.

[49] Luigi Catuogno, Alexandra Dmitrienko, Konrad Eriksson, Dirk Kuhlmann, Gianluca Ramunno, Ahmad-Reza Sadeghi, Steffen Schulz, Matthias Schunter, Marcel Winandy, and Jing Zhan. Trusted virtual domains: Design, implementation and lessons learned. In *Proceedings of the First International Conference on Trusted Systems*, INTRUST'09, pages 156–179, Berlin, Heidelberg, 2010. Springer-Verlag.

[50] E. Cayirci. A joint trust and risk model for msaas mashups. In *Simulation Conference (WSC), 2013 Winter*, pages 1347–1358, Dec 2013.

[51] A. Celesti, F. Tusa, M. Villari, and A. Puliafito. How to Enhance Cloud Architectures to Enable Cross-Federation. In *3rd IEEE International Conference on Cloud Computing (CLOUD)*, 2010.

[52] Cfengine. `http://www.cfengine.com`.

[53] CFEngine Community. cf-serverd brokend behavior in mixed IPv4/IPv6 environments. `https://dev.cfengine.com/issues/3873`, 2013.

[54] H. Chang, A. Hari, S. Mukherjee, and T. V. Lakshman. Bringing the Cloud to the Edge. In *IEEE INFOCOM 2014 Workshop on Mobile Cloud Computing (MCC)*, 2014.

[55] Chef. `http://www.opscode.com/chef/`.

[56] Liang Chen. *Analyzing and Developing Role-Based Access Control Models*. PhD thesis, Royal Holloway, University of London, 2011.

[57] P.M. Chen and B.D. Noble. When virtual is better than real [operating system relocation to virtual machines]. In *HotOS 2001*, 2001.

[58] D. M. Chess, C. Palmer, and S. R. White. Security in an Autonomic Computing Environment. *IBM Systems Journal*, 42(1):107–118, 2003.

[59] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

[60] Patrick Colp, Mihir Nanavati, Jun Zhu, William Aiello, George Coker, Tim Deegan, Peter Loscocco, and Andrew Warfield. Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2011.

[61] Criu. `http://www.criu.org/`.

[62] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. Remus: High availability via asynchronous virtual machine replication. In *5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 161–174, Berkeley, CA, USA, 2008. USENIX Association.

[63] Frederic Cuppens, Nora Cuppens-Boulahia, and Céline Coma. MotOrBAC : un outil d'administration et de simulation de politiques de sécurité. In *First Joint Conference on Security in Networks Architectures (SAR) and Security of Information Systems (SSI)*, 2006.

[64] Data Sealing, 2009. `Lecture Trusted Computing at RUHR-UNIVERSITY BOCHUM`.

[65] Juri Luca De Coi, Daniel Olmedilla, Piero Bonatti, and Luigi Sauro. Protune: A framework for semantic web policies. In Christian Bizer and Anupam Joshi, editors, *Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008)*, volume 401. CEUR Workshop Proceedings, 2008.

[66] Nathan Dimmock, András Belokosztolszki, David Eyers, Jean Bacon, and Ken Moody. Using trust and risk in role-based access control policies. In *Proceedings of the ninth ACM symposium on Access control models and technologies*, SACMAT '04, pages 156–162, New York, NY, USA, 2004. ACM.

[67] Docker. `https://www.docker.com/`.

[68] Docker Security, 2014. `https://docs.docker.com/articles/security/`.

[69] A. Eghtesadi, Y. Jarraya, M. Debbabi, and M. Pourzandi. Preservation of Security Configurations in the Cloud. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 17–26, March 2014.

[70] Marwa El Houri. *A Formal Model to express Dynamic Policies Access Control and Trust Negotiation a Distributed Environment.* PhD thesis, Université Toulouse III Paul Sabatier (UPS), 2010.

[71] A.A. El Kalam. Specification and enforcement of access control in information and communication systems. In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pages 1–6, April 2008.

[72] D. R. Engler, M. F. Kaashoek, and J. O'Toole, Jr. Exokernel: An Operating System Architecture for Application-level Resource Management. In *Fifteenth ACM Symposium on Operating Systems Principles*, SOSP'95, pages 251–266, New York, NY, USA, 1995. ACM.

[73] B. Kauer et al. Recursive Virtual Machines for Advanced Security Mechanisms. IEEE DSN Workshops, 2011.

[74] ETSI. Network Functions Virtualisation (NFV): Architectural Framework, 2014. ETSI Industry Specification Group ETSI GS NFV 002 V1.2.1.

[75] Hui Fang, Jie Zhang, Murat Sensoy, and Nadia Magnenat Thalmann. A generalized stereotypical trust model. In *Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, TRUSTCOM '12, pages 698–705, Washington, DC, USA, 2012. IEEE Computer Society.

[76] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, August 2001.

[77] D.F. Ferraiolo and D.R. Kuhn. Role-based access controls. *arXiv preprint arXiv:0903.2171*, pages 554 – 563, 2009.

[78] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, and B. Thuraisingham. Rowlbac: representing role based access control in owl. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, SACMAT '08, pages 73–82, New York, NY, USA, 2008. ACM.

[79] A. Fishman, Mike Rapoport, Evgeny Budilovsky, and Izik Eidus. HVX: Virtualizing the Cloud. In *HotCloud'13*, 2013.

[80] Liana Fong and Malgorzata Steinder. Duality of virtualization: simplification and complexity. *SIGOPS Oper. Syst. Rev.*, 42:96–97, January 2008.

[81] Afshar Ganjali and David Lie. Auditing cloud management using information flow tracking. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing*, STC '12, pages 79–84, New York, NY, USA, 2012. ACM.

[82] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: A Virtual Machine-Based Platform for Trusted Computing. *SIGOPS Oper. Syst. Rev.*, 37(5):193–206, 2003.

[83] Tal Garfinkel, Mendel Rosenblum, et al. A virtual machine introspection based architecture for intrusion detection. In *NDSS*, volume 3, pages 191–206, 2003.

[84] Valerio Genovese. *Modalities in Access Control: Logics, Proof-theory and Applications*. PhD thesis, University of Luxembourg and University of Torino, 2012.

[85] Google GCE. `https://cloud.google.com/`.

[86] Tyrone Grandison and Morris Sloman. Trust management tools for internet applications. In *Proceedings of the 1st international conference on Trust management*, iTrust'03, pages 91–107, Berlin, Heidelberg, 2003. Springer-Verlag.

[87] Stéphane Guilloteau and Venkatesen Mauree. Privacy in Cloud Computing, March 2012. ITU-T Technology Watch Report.

[88] S.M. Habib, S. Ries, and M. Muhlhauser. Towards a trust management system for cloud computing. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 933–939, Nov 2011.

[89] S. Hagen, M. Seibold, and A Kemper. Efficient verification of IT change operations or: How we could have prevented Amazon's cloud outage. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 368–376, April 2012.

[90] Sebastian Hagen. *Algorithms for the Efficient Verification and Planning of Information Technology Change Operations*. PhD thesis, Technische Universitt Mnchen, 2013.

[91] Fang Hao, T. V. Lakshman, Sarit Mukherjee, and Haoyu Song. Secure cloud computing with a virtualized network infrastructure. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 16–16, Berkeley, CA, USA, 2010. USENIX Association.

[92] Amir Herzberg, Haya Shulman, Johanna Ullrich, and Edgar Weippl. Cloudoscopy: Services discovery and topology mapping. In *Proceedings of the 2013 ACM Workshop on Cloud Computing Security Workshop*, CCSW '13, pages 113–122, New York, NY, USA, 2013. ACM.

[93] A Herzig and E Lorini. A logic of trust and reputation. *Logic Journal of IGPL*, 2010.

[94] Susan Hinrichs. Policy-Based Management: Bridging the Gap. In *Proceedings of the 15th Annual Computer Security Applications Conference*, ACSAC '99, pages 209–, Washington, DC, USA, 1999. IEEE Computer Society.

[95] Timothy L. Hinrichs, Natasha Gude, Martin Casado, John C. Mitchell, and Scott Shenker. Expressing and enforcing flow-based network security policies language. Technical report, University of Chicago, 2008.

[96] JF Hübner, Emiliano Lorini, Andreas Herzig, and Laurent Vercouter. From cognitive trust theories to computational trust. In *Proceedings of The 12th Workshop on Trust in Agent Societies*, 2009.

[97] Trung Dong Huynh, Nicholas R. Jennings, and Nigel R. Shadbolt. An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 13(2):119–154, September 2006.

[98] Ironic. `http://github.com/openstack`.

[99] Corinne S. Irwin and Dennis C. Taylor. Identity, credential, and access management at nasa, from zachman to attributes. In *Proceedings of the 8th Symposium on Identity and Trust on the Internet*, IDtrust '09, pages 1–14, New York, NY, USA, 2009. ACM.

[100] Keith Irwin and Ting Yu. An identifiability-based access control model for privacy protection in open systems. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, WPES '04, pages 43–43, New York, NY, USA, 2004. ACM.

[101] Daniel Jackson. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11:256–290, April 2002.

[102] Y. Jarraya, A. Eghtesadi, M. Debbabi, Y. Zhang, and M. Pourzandi. Cloud Calculus: Security verification in elastic cloud computing platform. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pages 447–454, May 2012.

[103] Yosr Jarraya, Arash Eghtesadi, Mourad Debbabi, Ying Zhang, and Makan Pourzandi. Formal Verification of Security Preservation for Migrating Virtual Machines in the Cloud. In *Proceedings of the 14th International Conference on Stabilization, Safety, and Security of Distributed Systems*, SSS'12, pages 111–125, Berlin, Heidelberg, 2012. Springer-Verlag.

[104] Audun Jøsang. Trust and reputation systems. In Alessandro Aldini and Roberto Gorrieri, editors, *Foundations of security analysis and design IV*, pages 209–245. Springer-Verlag, Berlin, Heidelberg, 2007.

[105] Audun Jøsang and Roslan Ismail. The Beta Reputation System. In *Proceedings of the 15th Bled Electronic Commerce Conference*, volume 160, pages 324–337, 2002.

[106] Lalana Kagal, Tim Finin, and Anupam Joshi. A policy based approach to security for the semantic web. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 402–418. Springer Berlin Heidelberg, 2003.

[107] A.A.E. Kalam, R.E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, and G. Trouessin. Organization based access control. In *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on*, pages 120–131, June 2003.

[108] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real Time Network Policy Checking Using Header Space Analysis. In *10th USENIX Symposium on Networked Systems Design and Implementation*, pages 99–111, Berkeley, CA, 2013. USENIX.

[109] Eric Keller, Jakub Szefer, Jennifer Rexford, and Ruby B. Lee. NoHype: Virtualized Cloud Infrastructure Without the Virtualization. In *37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 350–361, New York, NY, USA, 2010. ACM.

[110] Safwan Mahmud Khan, Kevin W. Hamlen, and Murat Kantarcioglu. Silver lining: Enforcing secure information flow at the cloud edge. In *Proceedings of the 2014 IEEE International Conference on Cloud Engineering*, IC2E '14, pages 37–46, Washington, DC, USA, 2014. IEEE Computer Society.

[111] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. VeriFlow: Verifying Network-Wide Invariants in Real Time. In *10th USENIX Symposium on Networked Systems Design and Implementation*, pages 15–27, Berkeley, CA, 2013. USENIX.

[112] S. Kikuchi and K. Hiraishi. Improving reliability in management of cloud computing infrastructure by formal methods. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–7, May 2014.

[113] Shinji Kikuchi. *Improving Reliability in Management of Cloud Computing Infrastructure by Formal Methods*. PhD thesis, Japan Advanced Institute of Science and Technology, 2013.

[114] Hyojoon Kim, Theophilus Benson, Aditya Akella, and Nick Feamster. The Evolution of Network Configuration: A Tale of Two Campuses. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 499–514, New York, NY, USA, 2011. ACM.

[115] A. Kivity, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti, and Vlad Zolotarov. OSv - Optimizing the Operating System for Virtual Machines. In *USENIX ATC'14*.

[116] F. John Krautheim. Private Virtual Infrastructure for Cloud Computing. In *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, HotCloud'09, Berkeley, CA, USA, 2009. USENIX Association.

[117] Yann Krupa. *PrivaCIAS: Privacy as Contextual Integrity in Decentralized Multi-Agent Systems*. PhD thesis, Ecole Nationale Supérieure des Mines de Saint-Etienne, 2012.

[118] Yann Krupa, Laurent Vercouter, Jomi Fred Hubner, and Andreas Herzig. Trust based evaluation of wikipedia's contributors. In Huib Aldewereld, Virginia Dignum, and Gauthier Picard, editors, *Engineering Societies in the Agents World X*, volume 5881 of *Lecture Notes in Computer Science*, pages 148–161. Springer Berlin Heidelberg, 2009.

[119] KVM. `http://www.linux-kvm.org/page/main_page`.

[120] Marc Lacoste, Aurélien Wailly, Aymeric Tabourin, Loïc Habermacher, Xavier Le Guillou, and Jean-Philippe Wary. Flying over Mobile Clouds with Security Planes: Select Your Class of SLA for End-to-End Security. In *6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, 2013.

[121] Bo Lang, Ian Foster, Frank Siebenlist, Rachana Ananthakrishnan, and Tim Freeman. A flexible attribute based access control method for grid computing. *Journal of Grid Computing*, 7(2):169–180, 2009.

[122] Adrien Lebre, Jonathan Pastor, and Mario Sudholt. VMPlaceS: A Generic Tool to Investigate and Compare VM Placement Algorithms. In *21st International European Conference on Parallel and Distributed Computing(Euro-Par)*, 2015.

[123] Adam J. Lee. *Towards Practical and Secure Decentraliz Attribute-Based Authorisation Systems*. PhD thesis, University of Illinois, 2008.

[124] Min Li, Wanyu Zang, Kun Bai, Meng Yu, and Peng Liu. Mycloud: Supporting user-configured privacy protection in cloud computing. In *Proceedings of the 29th Annual Computer Security Applications Conference*, pages 59–68. ACM, 2013.

[125] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, SP '02, pages 114–, Washington, DC, USA, 2002. IEEE Computer Society.

[126] Xin Liu, Anwitaman Datta, Krzysztof Rzadca, and Ee-Peng Lim. Stereotrust: a group based personalized trust model. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 7–16, New York, NY, USA, 2009. ACM.

[127] MAAS. `https://maas.ubuntu.com/`.

[128] A. Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels: Library Operating Systems for the Cloud. ASPLOS'13.

[129] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding BGP Misconfiguration. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, pages 3–16, New York, NY, USA, 2002. ACM.

[130] D.W. Manchala. Trust metrics, models and protocols for electronic commerce transactions. In *Proceedings of 18th International Conference on Distributed Computing Systems (Cat. No.98CB36183)*, pages 312–321. IEEE Comput. Soc, 1998.

[131] Stephen Paul Marsh. *Formalising trust as a computational concept*. PhD thesis, Department of Computing Science and Mathematics, University of Stirling, 1994.

[132] J. Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. ClickOS and the Art of Network Function Virtualization. NSDI'14.

[133] Jeanna Matthews, Tal Garfinkel, Christofer Hoff, and Jeff Wheeler. Virtual machine contracts for datacenter and cloud computing environments. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, ACDC '09, pages 25–30, New York, NY, USA, 2009. ACM.

[134] John McDermott, Bruce Montrose, Margery Li, James Kirby, and Myong Kang. Separation Virtual Machine Monitors. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 419–428, New York, NY, USA, 2012. ACM.

[135] Tom Mens, Juan Fernndez-Ramil, and Sylvain Degrandsart. The evolution of Eclipse. In *Proc. 24th Int'l Conf. on Software Maintenance*, 2008.

[136] Michael Menzel, Markus Klems, Hoàng Anh Le, and Stefan Tai. A configuration crawler for virtual appliances in compute clouds. In *Proceedings of the 2013 IEEE International Conference on Cloud Engineering*, IC2E '13, pages 201–209, Washington, DC, USA, 2013. IEEE Computer Society.

[137] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3), 2002.

[138] Yogesh Mundada, Anirudh Ramachandran, and Nick Feamster. Silverline: Data and network isolation for cloud services. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'11, pages 13–13, Berkeley, CA, USA, 2011. USENIX Association.

[139] Sanjai Narain. Network configuration management via model finding. In *Proceedings of the 19th conference on Large Installation System Administration Conference - Volume 19*, LISA '05, pages 15–15, Berkeley, CA, USA, 2005. USENIX Association.

[140] Sanjai Narain, Y.-H. Alice Cheng, Alex Poylisher, and Rajesh Talpade. Network single point of failure analysis via model finding. In *Proceedings of First Alloy Workshop*, 2006.

[141] Wolfgang Nejdl, Daniel Olmedilla, and Marianne Winslett. Peertrust: Automated trust negotiation for peers on the semantic web. In Willem Jonker and Milan Petković, editors, *Secure Data Management*, volume 3178 of *Lecture Notes in Computer Science*, pages 118–132. Springer Berlin Heidelberg, 2004.

[142] J. Nogueira, M. Melo, J. Carapinha, and S. Sargento. A distributed approach for virtual network discovery. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 277–282, Dec 2010.

[143] Talal H. Noor and Quan Z. Sheng. Trust as a service: A framework for trust management in cloud environments. In *Proceedings of the 12th International Conference on Web Information System Engineering*, WISE'11, pages 314–321, Berlin, Heidelberg, 2011. Springer-Verlag.

[144] Matunda Nyanchama and Sylvia Osborn. The role graph model and conflict of interest. *ACM Transactions on Information and System Security*, 2(1):3–33, February 1999.

[145] Openstack Consortium. AgregateMultiTenancyIsolation description incorrect. `https://bugs.launchpad.net/openstack-manuals/+bug/1328400`, 2014.

[146] Openstack Consortium. nova-manage creates network with wrong vlanid. `https://bugs.launchpad.net/nova/+bug/1288609`, 2014.

[147] OpenStack Consortium. Openstack. `http://www.openstack.org`, 2014.

[148] David Oppenheimer, Archana Ganapathi, and David A. Patterson. Why do internet services fail, and what can be done about it? In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, Berkeley, CA, USA, 2003. USENIX Association.

[149] Vasilis Pappas, VasileiosP. Kemerlis, Angeliki Zavou, Michalis Polychronakis, and AngelosD. Keromytis. Cloudfence: Data flow tracking as a cloud service. In SalvatoreJ. Stolfo, Angelos Stavrou, and CharlesV. Wright, editors, *Research in Attacks, Intrusions, and Defenses*, volume 8145 of *Lecture Notes in Computer Science*, pages 411–431. Springer Berlin Heidelberg, 2013.

[150] B.D. Payne, M.D.P. de Carbone, and Wenke Lee. Secure and flexible monitoring of virtual machines. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 385–397, Dec 2007.

[151] Siani Pearson. Towards Accountability in the Cloud. Technical Report HPL-2011-138, HP Laboratories, 2011.

[152] Diego Perez-Botero, Jakub Szefer, and Ruby B. Lee. Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers. In *International CLOUD Workshop on Security in Cloud Computing (SCC)*, 2013.

[153] Lucian Popa, Minlan Yu, Steven Y. Ko, Sylvia Ratnasamy, and Ion Stoica. Cloudpolice: Taking access control out of the network. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 7:1–7:6, New York, NY, USA, 2010. ACM.

[154] Christian Priebe, Divya Muthukumaran, Dan O' Keeffe, David Eyers, Brian Shand, Ruediger Kapitza, and Peter Pietzuch. Cloudsafetynet: Detecting data leakage between cloud tenants. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, CCSW '14, pages 117–128, New York, NY, USA, 2014. ACM.

[155] Puppet. `http://puppetlabs.com`.

[156] Himanshu Raj, Ripal Nathuji, Abhishek Singh, and Paul England. Resource management for isolation enhanced cloud services. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 77–84, New York, NY, USA, 2009. ACM.

[157] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212, New York, NY, USA, 2009. ACM.

[158] Sandra Rueda, Hayawardh Vijayakumar, and Trent Jaeger. Analysis of Virtual Machine System Policies. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, SACMAT '09, pages 227–236, New York, NY, USA, 2009. ACM.

[159] Jordi Sabater and Carles Sierra. Regret: reputation in gregarious societies. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 194–195, New York, NY, USA, 2001. ACM.

[160] Ahmad-Reza Sadeghi and Christian Stüble. Property-based Attestation for Computing Platforms: Caring About Properties, Not Mechanisms. In *Proceedings of the 2004 Workshop on New Security Paradigms*, NSPW '04, pages 67–77, New York, NY, USA, 2004. ACM.

[161] Reiner Sailer, Enriquillo Valdez, Trent Jaeger, Ronald Perez, Leendert Van Doorn, John Linwood Griffin, Stefan Berger, Reiner Sailer, Enriquillo Valdez, Trent Jaeger, Ronald Perez, Leendert Doorn, John Linwood, and Griffin Stefan Berger. sHype: Secure Hypervisor Approach to Trusted Virtualized Systems. In *IBM Research Report RC23511*, 2005.

[162] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The nist model for role-based access control: towards a unified standard. In *Proceedings of the fifth ACM workshop on Role-based access control*, RBAC '00, pages 47–63, New York, NY, USA, 2000. ACM.

[163] Ravi S. Sandhu. Lattice-based access control models. *Computer*, 26(11):9–19, November 1993.

[164] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-Based Access Control Models. *Computer*, 29(2):38–47, 1996.

[165] Nuno Santos, Krishna P. Gummadi, and Rodrigo Rodrigues. Towards Trusted Cloud Computing. In *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, HotCloud'09, Berkeley, CA, USA, 2009. USENIX Association.

[166] Nuno Santos, Rodrigo Rodrigues, Krishna P. Gummadi, and Stefan Saroiu. Policy-sealed Data: A New Abstraction for Building Trusted Cloud Services. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

[167] A. Scannel. NoVM: Hypervisor rebooted, 2014.

[168] Joshua Schiffman, Thomas Moyer, Hayawardh Vijayakumar, Trent Jaeger, and Patrick McDaniel. Seeding Clouds with Trust Anchors. In *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop*, CCSW '10, pages 43–46, New York, NY, USA, 2010. ACM.

[169] Joshua Schiffman, Yuqiong Sun, Hayawardh Vijayakumar, and Trent Jaeger. Cloud Verifier: Verifiable Auditing Service for IaaS Clouds. In *Proceedings of the IEEE 1st International Workshop on Cloud Security Auditing (CSA 2013)*, June 2013.

[170] Joshua Schiffman, Hayawardh Vijayakumar, and Trent Jaeger. Verifying System Integrity by Proxy. In *5th International Conference on Trust and Trustworthy Computing*, pages 179–201, 2012.

[171] Bianca Schroeder and Garth A. Gibson. A large-scale study of failures in high-performance computing systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 249–258, Washington, DC, USA, 2006. IEEE Computer Society.

[172] Ilari Shafer, Snorri Gylfason, and Gregory R. Ganger. vQuery: a Platform for Connecting Configuration and Performance. *VMware Technical Journal*, 1(2), December 2012.

[173] Seungwon Shin and Guofei Gu. CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–6, Oct 2012.

[174] Lenin Singaravelu, Calton Pu, Hermann Härtig, and Christian Helmuth. Reducing tcb complexity for security-sensitive applications: Three case studies. *ACM SIGOPS Operating Systems Review*, 40(4):161–174, 2006.

[175] S. Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors. EuroSys'07.

[176] A. Squicciarini, E. Bertino, Elena Ferrari, F. Paci, and B. Thuraisingham. Pp-trust-x: A system for privacy preserving trust negotiations. *ACM Trans. Inf. Syst. Secur.*, 10(3), July 2007.

[177] Udo Steinberg and Bernhard Kauer. NOVA: A Microhypervisor-based Secure Virtualization Architecture. In *5th European Conference on Computer Systems*, EuroSys'10, pages 209–222, New York, NY, USA, 2010. ACM.

[178] Lili Sun, Hua Wang, Jianming Yong, and Guoxin Wu. Semantic access control for cloud computing based on e-healthcare. In *Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on*, pages 512–518, May 2012.

[179] Bo Tang, Qi Li, and R. Sandhu. A multi-tenant rbac model for collaborative cloud services. In *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*, pages 229–238, July 2013.

[180] Bo Tang and R. Sandhu. Cross-tenant trust models in cloud computing. In *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, pages 129–136, Aug 2013.

[181] Bo Tang, R. Sandhu, and Qi Li. Multi-tenancy authorization models for collaborative cloud services. In *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, pages 132–138, May 2013.

[182] Mukarram Tariq, Amgad Zeitoun, Vytautas Valancius, Nick Feamster, and Mostafa Ammar. Answering What-if Deployment and Configuration Questions with Wise. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 99–110, New York, NY, USA, 2008. ACM.

[183] The Japan Times. Data on 5,700 firms lost by Yahoo unit. `http://www.japantimes.co.jp/news/2012/06/27/business/data-on-5700-firms-lost-by-yahoo-unit/`, 2012.

[184] Adel Nadjaran Toosi, Rodrigo N. Calheiros, and Rajkumar Buyya. Interconnected Cloud Computing Environments: Challenges, Taxonomy, and Survey. *ACM Comput. Surv.*, 47(1):7:1–7:47, May 2014.

[185] D. C. van Moolenbroek, R. Appuswamy, and A. S. Tanenbaum. Towards a Flexible, Lightweight Virtualization Alternative. SYSTOR'14.

[186] A. Vasudevan, Sagar Chaki, Limin Jia, Jonathan McCune, James Newsome, and Anupam Datta. Design, Implementation and Verification of an eXtensible and Modular Hypervisor Framework. IEEE SSP 2013.

[187] Anthony Velte and Toby Velte. *Microsoft Virtualization with Hyper-V*. McGraw-Hill, Inc., New York, NY, USA, 2010.

[188] Laurent Vercouter and Guillaume Muller. L.I.A.R.: Achieving Social Control in Open and Decentralized Multiagent Systems. *Applied Artificial Intelligence*, 24(8):723–768, September 2010.

[189] Aurélien Wailly, Marc Lacoste, and Hervé Debar. Towards Multi-Layer Autonomic Isolation of Cloud Computing and Networking Resources. In *Conference on Network and Information Systems Security (SARSSI)*, 2011.

[190] Aurélien Wailly, Marc Lacoste, and Hervé Debar. VESPA: Multi-Layered Self-Protection for Cloud Resources. In *International Conference on Autonomic Computing (ICAC)*, 2012.

[191] Yan Wang and Vijay Varadharajan. Role-based Recommendation and Trust Evaluation. In *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*, pages 278–288. IEEE, July 2007.

[192] Zhi Wang and Xuxian Jiang. Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 380–395, Washington, DC, USA, 2010. IEEE Computer Society.

[193] Jinpeng Wei, Xiaolan Zhang, Glenn Ammons, Vasanth Bala, and Peng Ning. Managing Security of Virtual Machine Images in a Cloud Environment. In *Proceedings of the ACM Workshop on Cloud Computing Security*, CCSW '09, pages 91–96. ACM, 2009.

[194] Dan Williams, Eslam Elnikety, Mohamed Eldehiry, Hani Jamjoom, Hai Huang, and Hakim Weatherspoon. Unshackle the Cloud! In *3rd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'11, pages 16–16, Berkeley, CA, USA, 2011. USENIX Association.

[195] Dan Williams, Hani Jamjoom, and Hakim Weatherspoon. The Xen-Blanket: virtualize once, run everywhere. In *Proceedings of the 7th ACM European Conference on Computer Systems*, pages 113–126. ACM, 2012.

[196] Dan Williams, Hani Jamjoom, and Hakim Weatherspoon. Plug into the Supercloud. *IEEE Internet Computing*, 17(2):28–34, March 2013.

[197] Timothy Wood, Alexandre Gerber, K. K. Ramakrishnan, Prashant Shenoy, and Jacobus Van der Merwe. The case for enterprise-ready virtual private clouds. In *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, HotCloud'09, Berkeley, CA, USA, 2009. USENIX Association.

[198] Ruoyu Wu, Gail-Joon Ahn, Hongxin Hu, and M. Singhal. Information flow control in cloud computing. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2010 6th International Conference on*, pages 1–7, Oct 2010.

[199] Reda Yaich, Olivier Boissier, Gauthier Picard, and Philippe Jaillon. Adaptiveness and Social-Compliance in Trust Management within Virtual Communities. *Web Intelligence and Agent Systems (WIAS), Special Issue: Web Intelligence and Communities*, 11(4), 2013.

[200] Walt Yao. Trust management for widely distributed systems. Technical Report UCAM-CL-TR-608, University of Cambridge Computer Laboratory, 2004.

[201] Younis A. Younis, Kashif Kifayat, and Madjid Merabti. An access control model for cloud computing. *Journal of Information Security and Applications*, 19(1):45 – 60, 2014.

[202] Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security*, 6(1):1–42, February 2003.

[203] Eric Yuan and Jin Tong. Attributed based access control (ABAC) for Web services. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE, 2005.

[204] Fengzhe Zhang, Jin Chen, Haibo Chen, and Binyu Zang. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 203–216, New York, NY, USA, 2011. ACM.

[205] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 305–316. ACM, 2012.

[206] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-tenant side-channel attacks in paas clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 990–1003, New York, NY, USA, 2014. ACM.

[207] Yinqian Zhang and Michael K Reiter. Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 827–838. ACM, 2013.