12
Real-Life Examples

This chapter outlines the main features of two of the most widely discussed Cloud management platforms: Amazon AWS (a commercial Cloud management platform) and OpenStack (an open-source Cloud management platform). Subsequently, the chapter presents a practical illustration of some of the concepts which are discussed throughout this book using OpenStack.

12.1 OpenStack

This section provides a high-level introduction to OpenStack.

12.1.1 What is OpenStack?

Chapter 3 presents an abstract view of Cloud management platforms, which we refer to as VCC. This section presents OpenStack [1] which is an example implementation of the VCC. OpenStack is an open-source software that was funded in October 2010 by Rackspace [2] and NASA [3]. Initially, OpenStack started by combining source codes from both RackSpace and NASA. A few months later, commercial companies started joining the OpenStack initiative. Currently, thousands of professionals

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

around the word participate in the OpenStack software architecture and code development process [1].

OpenStack's main objective is to establish a Cloud management platform that is capable of meeting the needs of the next generation of Cloud computing. OpenStack is designed to be the global Cloud trusted management platform, and it is not meant to replace a VMM or a hypervisor function. It does not even have an implementation of a hypervisor or a VMM, and rather implements a set of APIs that interact with different hypervisors running at the Cloud's physical servers. OpenStack is designed with the objective of being an independent hypervisor, by supporting a wide range of hypervisors. This would help in stopping vendor lock-in, which is a key requirement for the future success of Clouds, as discussed in Chapter 1. OpenStack currently supports the following hypervisors: KVM, QEMU, ESX, ESXi, and Xen.[1]

OpenStack is still under continuous development and is still missing many components such as logging, billing, and policy management. As a result, OpenStack plans releases on a six-month basis. Importantly, its components are still immature and do not cover many aspects of the capabilities of a potential Cloud. Such limitations make OpenStack, at the current time, not suitable for a production

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

environment (author's opinion). The main reason for the current limitations of Clouds is not related to OpenStack itself; it is rather related to the complexity and challenges of the potential Clouds which OpenStack aims to manage. OpenStack is supported by leading companies in their fields, such as IBM, vmware, Redhat, and many more.[2] Figure 12.1 lists the current releases of OpenStack.

Release name	Release date	OpenStack Compute version number	OpenStack Object Storage version number
Essex	April 2012	2012.1	1.4.8
Diablo	October 2011	2011.3	1.4.3
Cactus	April 2011	2011.2	1.3.0
Bexar	March 2011	2011.1	1.2.0
Austin	October 2010	0.9.0	1.0.0

Figure 12.1 Sample of OpenStack releases – a new release is added every six-months

12.1.2 Openstack Structure

In this section we briefly present the structure of OpenStack, as illustrated in Figure 12.2. It is composed of the following main components: identity service (also known as Keystone), compute service (also known as Nova), object storage service (also known as Swift), image service (also known as Glance), and dashboard service (also known as Horizon). More components are planned to be added in the future. This section briefly discusses these components.

Keystone. Keystone aims to provide an identity management service when interacting with

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

OpenStack. It covers the following functions: service management, user management, tenant management, and role management. Here, a tenant represents a group of resources (networks, volumes, instances, images, and keys) which belong to one project; a role represents an association of privileges, objects, and users.

Keystone manages access rights using RBAC [4], which grants access rights on a tenant's objects to users. Privileges are assigned to roles in service-specific files, which are currently stored at /etc/[SERVICE_CODENAME]/ policy.json. The identity service associates users with roles for tenants.

Nova. Nova[3] runs on physical servers to manage the resources which are allocated to servers. The resources which Nova manages include: networks, VMs, and volumes. The management of VMs covers interactions with the VMM to create, start, stop, and migrate VMs. Nova does not provide any virtualization capabilities by itself; instead, it uses an open-source library (e.g., libvirt APIs) to interact with the hypervisors as discussed previously.

Nova has the following additional components:

– Nova-api provides Cloud users with APIs to manage their Cloud resources.

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

– Nova-database is the central repository for OpenStack data (e.g., it stores compute resources, available volumes, and instances). More databases have been added recently, for example Glance and Keystone.

– Nova-schedule manages the hosting of VMs at the distributed servers of the Cloud infrastructure. Nova-scheduler allocates VMs to run on a physical computing node. The allocation of the physical computing node is based on the selected scheduling algorithm. Current supported schedulers include the following (see Figure 12.3):

Simple scheduler. This allocates VMs on physical servers by considering the least loaded host.

Chance scheduler. This allocates VMs randomly on physical servers.

Filter scheduler. This is a customized allocation of VMs on physical servers.

– Nova-network manages the network configurations of a VM. Nova-network supports the following types – in the current release of Openstack, only one type can be configured at a time:

Flat network. In this a VM is allocated a fixed IP address. The IP gets injected into the VM instance on launch.

DHCP network. In this a VM IP address is acquired by the VM instance from a

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

DHCP server running on the nova-network.

VLAN network. In this a switch is required to support VLAN tagging.

OpenStack configures bridges and virtual interfaces. Computing nodes would typically have two interfaces: public and internal interfaces. VM traffic to the outer world (i.e., between the internal interface and the public interface) is routed via the nova-network. The allocation of an IP address to a VM could either be a static IP (fixed for the instance lifetime) or a dynamic IP (that changes dynamically).

– Nova-volume manages the VM volumes (create/delete/attach/detach). Volumes provide persistent storage for use by instances.

– Message queue is a central hub for passing actions between OpenStack components.

Storage components. OpenStack supports two types of storage function: Swift and Nova-volume. Swift manages the storage of objects and provides scalable, reliable, and redundant backend storage. Swift is the storage option to consider when scalability and redundancy are required, but performance is not of concern. For example, backup and archival systems could be stored on Swift. On the contrary, Swift is not the right option for

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

continuously accessed data as in the case of an active database management system. Swift files are exposed through an HTTP interface.

Swift has the following components, as illustrated in Figure 12.4:

– *Swift proxy*. This intermediates the communication between Cloud users and their storage.

– *Swift object server*. This stores, retrieves, and deletes the Cloud user data in the form of binary large objects (blobs).

– *Swift container server*. This groups objects in containers (similar to the directory concept but without nesting).

– *Swift account server*. This is used to list containers.

– *RING*. This maps object names to their physical locations. Blobs, containers, and accounts have their own separate rings.

Nova-volume (or simple volume) is a detachable block storage device. The data stored on a volume is persistent even after instance deletion. A volume can be associated with only one VM instance at a time. Sharing file systems across instances should be provided using other mechanisms, such as NFS [5]. Nova-volume provides a block storage file system which is exposed through a low-level bus interface such as SCSI. Clients

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

access the storage when the storage devices are mounted on their virtual machines.

The Nova-volume service works as follows:

– A volume group named nova-volume should first be created using the command line 'nova volume-create,' which creates a logical volume in the nova-volume group.

– After successful execution of the previous command, the compute node, which holds the volume group, would have a new logical volume as local storage. This logical volume can now be attached to a specific instance running on the compute node.

– The logical volume can be created and attached to an instance using the following commands:

nova volume -create -display_name VOLUME_NAME SIZE_GB

nova volume-attach INSTANCE_ID VOLUME_ID

DESTINATION_DEVICE_MOUNT_POINT

Logical volumes could be configured as bootable volumes.

Glance component. Glance is an OpenStack image service which could be used to store and retrieve VM images (VMIs). Nova fetches the image to the hosting server and then boots it up on its host physical server. A VMI could also be managed by Swift, in which Nova follows

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

similar processes when fetching the image to be booted up on its host physical server. VMI could be created using either third-party tools (e.g., Oz, VMBuilder, and VeeWee) or manually as follows:

– Create an empty image using the command

kvm-img create -f raw/qcow IMAGE_NAME SIZE

– Get the OS ISO format and install it in the VMI using the commands

kvm -m SIZE -cdrom ISO_PATH drive file=IMAGE_NAME,if=virtio,index=0 -boot d -net nic -net user -nographic -vnc VNCDISPLAY -monitor unix:MONITOR_FILE _NAME,server,nowait

– Connect to the VMI instance using the VNC client and proceed with the installation instructions.

Once a VMI is created, it can be managed as follows:

– When using glance, upload the image to glance as follows:

glance image-create name="NAME_IN_GLANCE" is_public=true container_format =ovf disk_format=raw/qcow2 < IMAGE_NAME

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

– ‘nova image-list’ is one of the commands that lists images.

– A key pair could be created for accessing an image as follows:

nova keypair-add KEY_HANDLE_NAME > PUBLIC_KEY_FILE_NAME

– Booting nova image can be done as follows:

nova boot –image IMAGE_NAME –flavor m1.small –key_name KEY_HANDLE _NAME INSTANCE_NAME

– Deleting an instance can be done as follows:

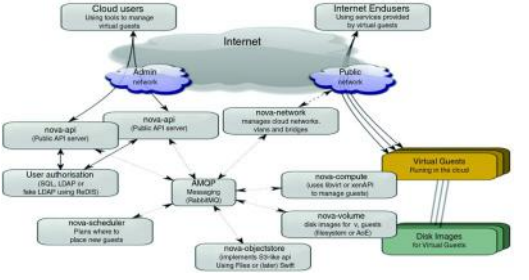
nova delete INSTANCE_NAME

– Pausing/suspending instances can be done as follows:

nova pause/unpause INSTANCE_NAME_ID

– Suspend/resume instances can be done as follows:

nova suspend/resume INSTANCE_NAME_ID



Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

Figure 12.2 Current OpenStack structure. *Source:* http://docs.openstack.org/

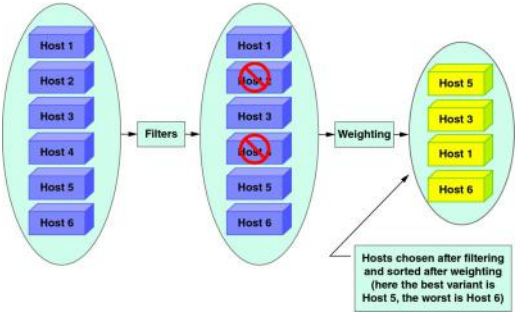


Figure 12.3 Current OpenStack scheduler. *Source:* http://docs.openstack.org/

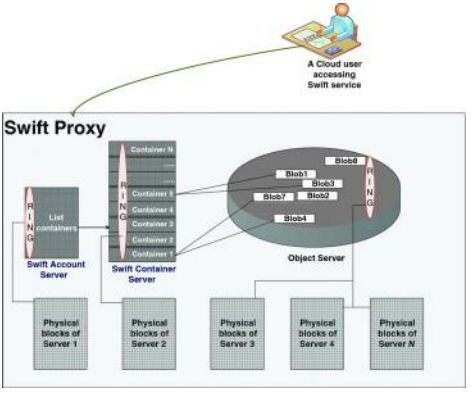


Figure 12.4 The main components of OpenStack Swift

12.1.3 Security in OpenStack

As discussed earlier, OpenStack supports identity management using the Keystone service. It also manages access rights using the RBAC mechanism. OpenStack uses the concept

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

of security groups to provide inbound network traffic filtering for instances. A security group is a collection of rules in an IP table that gets applied to the incoming packets of instances. Each security group can have multiple rules associated with it. Each rule specifies the source IP, protocol type, destination ports, etc. Only packets matching the rule are allowed in. A security group that does not have any rules associated with it causes blocking of all incoming traffic. A security group is attached to an instance on start-up. Outbound network traffic filtering, in contrast, needs to be implemented from inside VM instances.

OpenStack supports what is called 'availability zones.' Availability zones help in supporting application higher availability and resilience. They ensure the physical independence of redundant application resources when assigned to different availability zones. Physical independence could mean separate power supply, network equipment, physical location, etc.

OpenStack has implemented the remote attestation principle of TCG specifications, as discussed in Section 6.2. The implementation of the remote attestation principle allows users to specify the trust level of servers when hosting their resources.

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

12.1.4 OpenStack Configuration Files

OpenStack configurations are managed via text-based files. The files are stored in the following directory: /etc/SERVICE_NAME/ SERVICE_NAME.conf, where SERVICE_NAME is a variable representing the name of an OpenStack service. For example, the nova configuration file is stored by default in /etc/nova/nova.conf and it has the following type of configuration parameters: # LOGS, # AUTHENTICATION, # SCHEDULER, # VOLUMES, # DATABASE, # COMPUTE, # APIS, # GLANCE, and # NETWORK. OpenStack services run by a set of daemons that have a name starting with the service name, such as nova-*, glance-*, keystone-*, etc. They could either run on a single machine or be spread across multiple machines.

12.2 Amazon Web Services

Amazon Web Services (AWS) is a Cloud management platform that provides computing resources and services on a pay-per-use model. Amazon AWS and OpenStack have some comparable features. We could say that OpenStack provides almost all services that AWS provides and, in addition, OpenStack provides additional features that AWS cannot cover. This is because AWS targets a production environment, that is it can only support stable and proven-to-work technology that has been

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

tested to work reliably in critical infrastructures.

OpenStack, in contrast, is an open-source research-oriented project aiming to establish the next generation of trustworthy Cloud computing. Unlike AWS, OpenStack as a result is still considered not production-ready. We now list the most important services in AWS and how they map to those of OpenStack.

Amazon Elastic Compute Cloud (Amazon EC2) provides computing resources in AWS. It is somewhat similar to Nova compute in OpenStack. AWS uses different naming conventions from those of OpenStack; for example, a VMI in AWS is called an Amazon machine image (AMI) and a VM instance is called an Amazon EC2 instance.

Amazon supports three types of storage systems, as illustrated in Figure 12.5: Amazon Elastic Block Store (Amazon EBS), Amazon Simple Storage Service (Amazon S3), and Amazon EC2 instance store. All these storage systems are available at OpenStack. Amazon EBS is persistent storage, which is similar to Nova-volume in OpenStack. Such volumes are not affected by an instance lifecycle. That is, if the instance is terminated for any reason, then its attached Amazon EBS volumes keep the data intact. OpenStack associates with each instance a local storage which is deleted when its instance is terminated. This is similar to the

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

Amazon EC2 instance store. Amazon S3 is similar to the OpenStack Swift component. It provides access to a reliable and inexpensive data storage infrastructure.

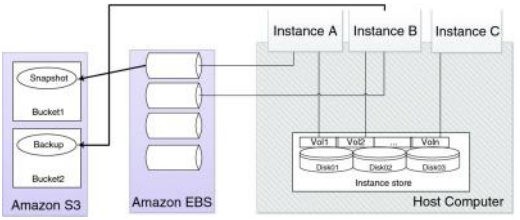


Figure 12.5 Amazon storage systems

The security measures provided by Amazon are also covered by OpenStack. OpenStack Keystone is called Identity and Access Management (IAM) by AWS. IAM manages users, federated users, and roles. AWS is also similar to OpenStack when controlling access to network traffic. Users define appropriate firewall rules controlling network traffic to their VMs. In addition, AWS as in OpenStack supports availability zones to split redundant resources across separate physical resources. As indicated earlier, OpenStack provides richer features than those provided in AWS. For example, unlike OpenStack, AWS does not yet support the assessment of servers' trustworthiness.

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

12.3 Component Architecture

Figure 12.6 presents a high-level architecture which illustrates the main entities and a general layout of a scheme framework that implements some of the concepts discussed throughout the book. We use OpenStack controller node (i.e., the VCC) and OpenStack nova-compute (i.e., a computing node at the physical layer). The computing node runs a hypervisor which manages a set of VMs. The VCC receives two main inputs: user requirements and infrastructure properties. The VCC manages user virtual resources based on such an input. This section introduces new components to OpenStack. Adding a new component to OpenStack requires updating the following components: *nova-api*, *nova-database*, *nova-scheduler*, and *nova-compute*. In this section we present the modifications that are introduced on these components.

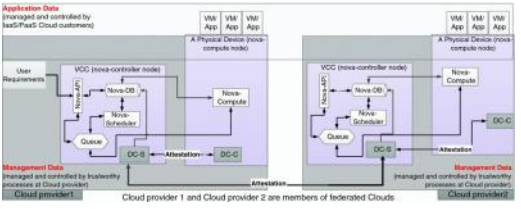


Figure 12.6 High-level architecture

12.3.1 Nova-api

Nova-api is a set of command lines and graphical interfaces which are used by Cloud

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

customers when managing their resources at the Cloud, and are also used by Cloud administrators when managing the Cloud virtual infrastructure. We updated *nova-api* library to consider the following:

Infrastructure properties. The Cloud physical infrastructure is very well organized and managed, and its organization and management associate its components with infrastructure properties. Examples of such properties include: a resource chain of trust, components reliability and connectivity, components distribution across Cloud infrastructure, redundancy types, servers clustering and grouping, and network speed.

User requirements. These include technical requirements, service level agreement, and user-centric security and privacy requirements.

Changes. These represent changes in user properties (e.g., security/privacy settings), infrastructure properties (e.g., components reliability, components distribution across the infrastructure, and redundancy type), and infrastructure policy. The main changes which we introduced at *nova-api* include the following:

– Add an option to enable users to manage their requirements, which include but are not limited to security and privacy aspects. We tested our prototype on the following user requirements: geographical location and user

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

isolation, which are managed manually at this stage. The first controls the hosting location of user resources, while the second controls the exclusion of certain users from sharing a physical server. It is important to stress that (at this stage) we assume system administrators are trusted. Planned future work will provide a more stringent mechanism to eliminate such an assumption.

– Add an option which enables administrators to manage Cloud infrastructure properties and policies, for example associate computing nodes with their domains and collaborating domains.

– Provide an interface which enables automated collection of the properties of the physical resources through trustworthy channels – at this stage we focus specifically on automating the collection of RCoT.

The main changes which are related to this point include adding an option to enable users to add a wide range of requirements and manage them; that is, updating nova-manage to enable users to create, list, and change requirements, and also updating the same library (nova-manage) to enable the association with each physical resource of a set of properties using the command lines options add_properties, get_properties, and remove_properties as illustrated in Figure 12.7. These libraries communicate with

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

nova-database to store user choices and infrastructure properties, as discussed next. The stored data in *nova-database* is used by the presented scheduler ACaaS (access control as a service), as discussed next.

Figure 12.7 Nova-api updates

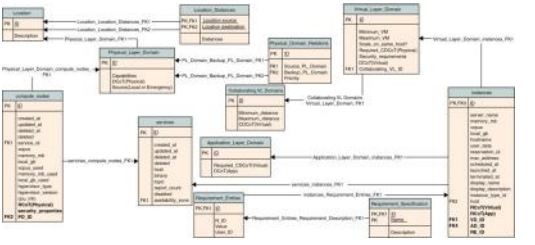


Figure 12.8 Updates on nova-database to include part of the identified Cloud relations

12.3.2 Nova-database

Nova-database is composed of many tables holding details of the Cloud components. It also holds users, projects, and security details (these get changed between different releases of

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

OpenStack). We extended *nova-database* in different directions to maintain the taxonomy of Clouds, user requirements, and infrastructure properties. Figure 12.8 illustrates the modifications at *nova-database* in bold format, which are as follows:

Compute_nodes is an existing nova-database table that holds records reflecting a computing resource at the physical layer. We updated this table by adding the following fields: RCoT(Physical) and security properties which hold a list of computing resource security details.

Requirement_entries and requirement_specification are two new tables. The first holds the expected type of requirement a user is allowed to enter, such as federated Cloud, location, excluded user list, etc. The second holds a user-specific value of the defined requirements. Figure 12.7 demonstrates user interactions with these tables via the nova-manage command line interface.

Physical_Layer_Domain is a new table which holds the records of Cloud physical domains. This covers both the primary Cloud provider physical domains and all other physical member domains of the emergency domain. The table defines the relationship amongst resources and holds physical domain metadata. The metadata includes the domain capabilities, DCoT, and a foreign key pointing to the table

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

which identifies the relative geographical location of the physical domain within the Clouds and federated Clouds infrastructure. The table has a field source, which can have two values: local or emergency. Emergency means it is a federated Clouds domain while local means it belongs to the same Cloud infrastructure.

Location and Location_Distances. The aim of these tables is to identify all possible *locations* at the Cloud infrastructure. They also define the relative distance between pairs of all identified locations. These tables are bound as follows: the compute_node table is bound to the physical_layer_domain table and the physical_layer_domain table is bound to a specific location identifier in the *location* table. The latter is bound to the location_distances table, which specifies all distances between a location identifier and all other location identifiers. In this we assume the resources of a physical domain are within close physical proximity, which reflects current deployment scenarios in practice.

Collaborating_PL_Domain is a new table which establishes the concept of collaborating physical domains. Each record identifies a specific backup domain for each physical domain with a priority value. A source domain can have many backup domains. The value of the priority field identifies the order in which physical backup domains could possibly be

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

allocated to serve as source domain needs. Backup domains are used in maintenance windows, emergencies, load balancing, etc. Backup domains should have the same capabilities and DCoT as the source physical domain itself.

Instances is an existing OpenStack table representing the running instances at computing nodes. We updated the table by adding the following fields: virtual resource chain of trust RCoT(Virtual); application resource chain of trust RCoT(Application); two foreign keys which establish a relationship with the instance's virtual and application domain tables, as defined in the *Virtual_Layer_Domain* and *Application_Layer_Domain* tables, respectively; and RS_ID which is a foreign key pointing to the requirement_entries table.

Services table is an existing OpenStack table which binds the virtual layer resources to their hosting resources at the physical layer.

Other tables. Openstack has many more tables, which are beyond the scope of this chapter to discuss.

Most of the *nova-database* records are uploaded automatically, using the software agents as discussed in Chapter 9, the modified *nova-api*, and/or via OpenStack management tools. Ideally, such records should be securely

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

collected and managed. At this stage our focus is on providing high-level architecture design, providing a running Cloud scheduler, and providing software agents that can attest to the trustworthiness of OpenStack components and then push the result to the nova-database. Full automation of Cloud management services is our planned long-term objective.

12.3.3 Nova-scheduler

In OpenStack, *nova-scheduler* controls the hosting of VMs at physical resources considering user requirements and infrastructure properties. Current implementations of *nova-scheduler* do not consider the entire Cloud infrastructure, nor do they consider the overall user and infrastructure properties. According to OpenStack documentation, nova-scheduler is still immature and great efforts are still required to improve it. We implement a new scheduler algorithm, ACaaS, which performs the following when allocating physical resources to host virtual resources: considers the discussed Cloud taxonomy; selects a physical domain's resource which has physical infrastructure properties that can best match user properties; and ensures that the user requirements are continually maintained. ACaaS collaborates with the following software agents (see Figure 12.6):

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

Cloud client agent, DC-C. Runs at OpenStack computing nodes and performs the following: calculates the computing node RCoT and continually assesses the status of the computing node and passes the result over to DC-S; manages domains and collaborating member domains based on policies distributed by DC-S (e.g., a VM can only operate with a known value of a chain of trust and when the hosting physical collaborating domains have a specific value of CDCoT(Physical) as defined by user properties).

Cloud server agent, DC-S. Runs at OpenStack domain controller and performs the following: maintains and manages OpenStack components (including the nova-scheduler) by ensuring they operate the Cloud only when they are trusted to behave as expected; manages the membership of the physical and virtual domains; and attests to DC-C trustworthiness when its computing node joins a physical domain. DC-S also intermediates the communication between DC-C and nova-scheduler, attests to DC-C's computing-node trustworthiness, collects the computing node RCoT, and then calculates DCoT, CDCoT, and stores the result in an appropriate field in nova-database.

12.4 Prototype

Having defined a high-level architecture of the scheme, this section describes a possible

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

prototyping. We present a mechanism for a trustworthy collection of resource chains of trust, and then calculate for each group of resources their domain and collaborating domain chain of trust. Subsequently, we use the ACaaS scheduler to match user properties with infrastructure properties. Other infrastructure properties are either collected automatically (such as the capabilities of physical resources) or entered manually (such as the physical location of computing nodes). These properties could be altered by system administrators. The trust measurements performed by the DC-C identify the building up of a resource's chains of trust and its integrity measurements. This section discusses the implementation of the scheme framework. Our implementation also includes trust establishment building on remote attestation and secure scheduling.

12.4.1 Trust Attestation via the DC-C

The implementation is based on an open-source trusted computing infrastructure which is built on a Linux operating system. Building the resource chain of trust of a computing node starts from the node TPM and ends with the node DC-C, as illustrated in Figure 12.9. The RCoT building process starts with the platform bootstrapping procedure, which initializes the TPM via a trusted BIOS. The trusted BIOS measures and loads the trusted bootloader [6] (i.e., the Trusted Grub), which measures and

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

loads a Linux kernel. We updated the Linux kernel ensuring that the IBM Integrity Measurement Architecture (IMA) [7] is enabled by default. The IMA measures all critical components before loading them. These include kernel modules, user applications, and associated configuration files. The values of these measurements are irreversibly stored inside the PCRs of the computing node, which are protected by the TPM. The IMA by default uses PCR #10 to store the measurements.

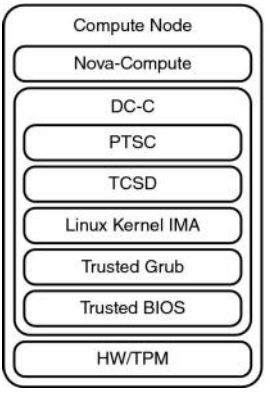


Figure 12.9 Compute node architecture

The TPM driver and the Trusted Core Service Daemon (TCSD) [8] expose the Trusted Computing Services (TCS) to applications. These components constitute the part of the DC-C for collecting and reporting the trust measurement of a resource. The resource chain of trust is, hence, constructed from the CRTM [9], which itself resides in and is protected by the trusted BIOS.

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

Table 12.1 illustrates part of the records of the bootstrapping process for the prototype as generated by the IMA measurement log. The IMA measurement log is the source for generating integrity reports (IRs), which are used, as we discuss later, to determine the genuine properties of a target system during the remote attestation process. The first column in Table 12.1 shows the value of *PCR*10 after loading the components of the third column. The second column records the hash value of the loaded component. The first record holds the value of *boot_aggregate*, which is a combined hash value of *PCR*0 – *PCR*7; that is, it possesses the measurement of the trusted computing base (TCB) of a computing node, including the trusted BIOS, the trusted bootloader, and the image of the Linux kernel together with its initial ram-disk and kernel arguments. Whenever a software component is loaded, the IMA module generates a hash value of the loaded component and then *extends* it into *PCR*10 by invoking the *TPM_Extend* command [10]. Such a command updates *PCR*10 to reflect the loaded component as follows: $pVal_i = hash(pVal_{i-1}, hVal_i)$. Subsequent rows in the table present the measurement logs of the bootstrapping workflow at the adopted operating system, Ubuntu 11.04. Other OpenStack components are then measured, which include the *nova-compute.conf* script, the *python*

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

executable, the *nova-compute* executable, supporting libraries, and critical configuration files.

Table 12.1 Compute node bootstrapping measurement log

***PCR*10**	**HASH**	**Loaded component**
$pVal_0$	$hVal_0$	boot_aggregate
$pVal_1$	$hVal_1$	/init
$pVal_2$	$hVal_2$	ld-linux-x86-64.so.2
$pVal_3$	$hVal_3$	libc.so.6
...	...	...
$pVal_i$	$hVal_i$	nova-compute.conf
...	...	...
$pVal_j$	$hVal_j$	python
...	...	...
$pVal_k$	$hVal_k$	nova-compute
$pVal_{k+1}$	$hVal_{k+1}$	libssl.so.1.0.0
...	...	...
$pVal_l$	$hVal_l$	nova.conf
...	...	...

To reduce the complexity and focus on a practical Cloud deployment case, the prototype turns off all unnecessary services at the base system. As a result, the value of *PCR*10 does not change except if a new software module (e.g., a

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

user program, kernel modules, or shared libraries) is loaded on the computing node. The loading process could be either good (e.g., a security patch) or malicious. In this case, the loaded software module would be measured and added to the log records. Such a measurement changes the value of *PCR*10.

Our prototype intentionally filters out the IMA measurements of VMs, that is the QEMU program in the prototype. This is because a VM CoT should be built on a compositional CoT; that is, the IMA measurements of a VM should not be considered as part of the TCB of a computing node. The measurements of a VM should rather be controlled by the IaaS Cloud user and not the Cloud provider, as this will likely raise the user's privacy concerns. This measurement process, in addition, would significantly increase the complexity of the trust management. If an exploited VM runs on a computing node, for example, to perform a malicious behavior on other components and applications, the properties of the computing node would change once the exploited VM started to affect the TCB components. In such a case, the DC-C will leave the physical domain; that is, the DC-C will stop operating and the VMs which are hosted at the infected computing node will be forced to migrate to another healthy computing node member of the same physical domain.

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

Finally, the DC-C collects the integrity measurement logs as recorded by the IMA, and generates an IR following the specifications of the platform trust service (PTS) interface [11]. The DC-C, as we discuss in the next subsection, sends the IR and the signed PCR values to the DC-S on request. In the prototype, this component is implemented by integrating the PTSC module from the OpenPTS [12].

12.4.2 Trust Management by the DC-S

This section starts by summarizing the high-level steps of the implemented part of the system workflow.[4] It then presents the prototyping details which are related to the DC-S. These are as follows:

Cloud security administrators could either create a new physical domain or use an existing domain. The creation process involves deciding on the domain capabilities, location, and defining its collaborating domains. As discussed in Section 12.3, we updated *nova-api* to enable administrators to manage this process.

Cloud security administrators then install the DC-C and nova-compute at all new physical computing nodes that are planned to join the domain.

The DC-C joins the cloud physical domain by communicating with the DC-S. The DC-S would first attest to the DC-C's trustworthiness and establish an offline chain of trust with the DC-C

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

(using sealing and remote attestation concepts, as proposed by TCG specifications). Next, the DC-C would calculate its host chain of trust RCoT, as described in Chapter 9, and pass the results to the DC-S.

Subsequently, the DC-S would store the RCoT at the *compute_nodes* table, and ensure that all devices in each domain have the same capabilities. The sealing mechanism, which is established in previous steps, assures the DC-S that the DC-C can only operate with the same value of the reported RCoT. If this value changes (e.g., as in the case of the hosting device being hacked), the DC-C will not operate. This prevents VMs from starting at a hacked device.

Users, using *nova-api* commands, deploy their VMs and associate them with certain properties. Such properties include, for example, the required CDCoT(Physical) and the multi-tenancy restrictions which control the sharing of a computing node with other users.

The ACaaS scheduler allocates an appropriate physical domain to host a user VM. The properties of the physical domain and its member devices should satisfy the defined user requirements.

The remaining part of this section covers the implementation of the remote attestation process and the secure scheduling.

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

Remote Attestation

Our prototype implements the remote attestation process using OpenPTS [12] which is managed by the DC-S. OpenPTS sends an attestation request to each computing node to retrieve its IR and PCR values. When a computing node sends the requested values, OpenPTS would then examine the consistency of the received IR and PCR values [9]. Subsequently, it would verify the security properties of the computing node by matching the reported IR with the expected measurement from a white-list database [9]. The white-list database stores sets of measurements, where each set is calculated based on a carefully selected *good* platform configuration state. The calculation is performed on a *good* platform in the form of hash values for a selected set of pre-loaded software components.

For the purpose of the prototype, we used two newly installed Ubuntu 11.04 servers. These servers have minimal settings for a computing node to perform its planned functions. The hash values of the software stack of each computing node should exist within the white-list database. If it does not, we consider the computing node to be untrusted. The *good* configurations could be extended/changed by adding/updating their corresponding values in the white-list database.

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

The attestation protocol works as follows. Every computing node (C_i) is identified by its AIK. The AIK is certified by the Cloud controller VCC (M) as it covers the Privacy-CA role [9, 14]. When a new computing node is added to the Cloud infrastructure it must first be registered at the VCC which then certifies its AIK. Only registered computing nodes can connect to the VCC as their certified AIKs cannot be forged and AIKs can only be used inside the genuine TPM that generates them. The registration steps of C_i at M are outlined in Protocol 12.1. Whenever a computing node sends a request to connect to the VCC a trust establishment protocol is executed which is outlined in Protocol 12.2.

Protocol 12.1 Computing Node Registration Protocol

A computing node (C_i) sends a registration request to VCC (M) as follows. First, C_i sends a request to its TPM to create an AIK key pair using the command TPM_CreateAIK. The TPM would then generate an AIK key pair. The generated private part of the key pair never leaves the TPM, and the corresponding public part of the key pair is signed by the TPM endorsement key (EK) [9]. The EK is protected by the TPM, and never leaves it. C_i then sends a registration request to M. The

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

request is associated with the EK certificate, the AIK public key, and other parameters:

$$C_i \rightarrow M : Cert(K_{EK_i}), \{K_{AIK_i}\}_{K_{EK_i}^{-1}} \quad (12.1)$$

M certifies AIK_i as follows. *M* verifies $Cert(EK_i)$. If the verification succeeds, *M* generates a specific-AIK certificate for C_i and a unique ID, CID_i. It then sends the result to C_i:

$$M \rightarrow C_i : \{Cert(K_{AIK_i}), CID_i\}_{K_M^{-1}}, Cert(K_M) \quad (12.2)$$

Protocol 12.2 Trust Establishment Protocol

M sends an attestation request to C_i. The request includes a nonce N_a. C_i would then report an attestation ticket to *M* as follows. C_i sends its PCR values, and the measurement log IR back to *M*, together with N_a. These are signed using the C_i's AIK:

$$C_i \rightarrow M : \{N_a, \{PCR\}, IR\}_{K_{AIK_i}^{-1}} \quad (12.3)$$

M then verifies the message sent by C_i as follows. It verifies the AIK_i signature and N_a

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

matches the sent nonce. If the verification succeeds, *M* examines the consistency of PCR and IR, and then determines the properties of C_i based on the value of IR.

The configurations of a computing node could possibly be altered after an attestation session, for example loading a new application. In such a case, the computing node attestation properties (as maintained by the VCC) would be violated. Addressing this would require establishing a trusted channel [15] to *seal* [9] the communication key with the verified PCR values. The sealing process provides the assurance that the DC-S can load the key only for a specific computing node's configuration. Any changes in the computing node configurations would trigger a new attestation request from the VCC to the computing node.

The implementation of the trusted channel, when sealed keys are loaded into memory, requires a small TCB. The TCB should enforce strict access to the memory area which stores the key. Having a large TCB, however, could result in leaking the key from memory without reflecting that on the platform trust status. Implementing a small TCB is a challenging problem, especially considering the complexity and scalability of the hosting Cloud system (we leave this important subject as planned future research). As an attempt to lessen the impact of

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

this threat, in the prototype we impose periodic attestations which keep the security properties of a computing node up to date. We implemented this by associating a timer with each computing node. Re-attestation is enforced whenever the timer expires. Untrusted computing nodes found by the re-attestation will be removed immediately from the database and would need to re-enroll in the system for future use. In addition, VMs running on untrusted computing nodes will be forced to migrate to other computing nodes which are members of the same physical domain.

Secure Scheduling

As we discussed in previous sections, computing nodes are organized into physical domains. Such organization is based on the properties of each computing node (i.e., security, privacy and other properties) which enable it to serve the needs of the domain. Users can specify their expected properties of computing nodes that could host their VMs. Some of the properties could be represented by a set of PCR values. However, PCR values are hard to pre-calculate and manage as they represent aggregated hash values of software components when loaded in a specific order. In the prototype, users do not need to specify PCR values, rather they would need to identify their desired hosting environment using the provided sets of white-lists. A computing node white-list

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

is identified in accordance with its properties which get attested whilst joining a domain and periodically thereafter. Genuine updates on the properties of a computing node (e.g., applying a security patch) require adjustment of the corresponding record in the white-list database. In the prototype, part of the user's required properties could represent entries in the white-list database. The ACaaS scheduler deploys each VM on a computing node that has the same properties as the one requested by the user of the VM. The ACaaS scheduler, in collaboration with the DC-S and DC-C, periodically examines the consistency of such properties.

12.5 Summary

Establishing the next generation of trustworthy Cloud infrastructure is a complex mission which requires collaborative efforts between industry and academia. This book presents the foundation of Cloud computing science building on solid in-depth and diverse experience in this domain. Part Two presented a set of integrated frameworks which form the roadmap for establishing trust in Clouds. It also presented a list of framework requirements and discussed possible solutions to some of the requirements. This chapter has presented a possible implementation of some components of such frameworks. We also introduced commercial

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

and open-source Cloud management platforms: Amazon AWS and OpenStack.

Notes

[1] More hypervisors are planned to be added in the future; for an up-to-date list, see http://wiki.openstack.org/HypervisorSupportMatrix.

[2] A full list of OpenStack partners can be found at: http://www.openstack.org/foundation/companies/.

[3] Also called Nova-compute.

[4] Further details about these steps are provided in [13].

References

OpenSource. OpenStack, 2010. http://www.openstack.org/.

RackSpace, 2013. http://www.rackspace.com.

NASA, 2013. http://www.nasa.gov.

R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control modles. In *IEEE Computer*, pp. 38–47. IEEE, 1996.

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

Sun Microsystems Inc. NFS: Network File System Protocol specification. RFC 1094, Internet Engineering Task Force, March 1989.

Trusted Grub. http://trousers.sourceforge.net/grub.html.

Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th Conference on USENIX Security Symposium – Volume 13, SSYM'04*, pp. 16–16. USENIX Association: Berkeley, CA, 2004.

Trousers – the open-source TCG software stack. http://trousers.sourceforge.net/.

Trusted Computing Group. http://www.trustedcomputinggroup.org.

Trusted Computing Group. *TPM Main, Part 3, Commands. Specification version 1.2 Revision 103*, 2007.

Infrastructure Work Group Platform Trust Services Interface specification, version 1.0. http://www.trus tedcomputinggroup.org/ resources/ infrastructure_work_group_platform_trust_services_interf _version_10, 2006.

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.

Open Platform Trusted Service User's Guide. http://iij.dl.sourceforge.jp/openpts/51879/userguide-0.2.4.pdf, 2011.

Imad M. Abbadi. Clouds infrastructure taxonomy, properties, and management services. In Ajith Abraham, Jaime Lloret Mauri, John F. Buford, Junichi Suzuki, and Sabu M. Thampi (eds), *Advances in Computing and Communications*, vol. 193 of *Communications in Computer and Information Science*, pp. 406–420. Springer-Verlag: Berlin, 2011.

Privacy ca. http://www.privacyca.com.

Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, and N. Asokan. Beyond secure channels. In *Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing, STC '07*, pp. 30–40. ACM: New York, 2007.

Copyright © 2014. John Wiley & Sons, Incorporated. All rights reserved.

Abbadi, IM 2014, Cloud Management and Security, John Wiley & Sons, Incorporated, New York. Available from: ProQuest Ebook Central. [26 September 2021].
Created from ucsl on 2021-09-26 12:57:52.