A framework for cloud assurance and transparency based on continuous evidence collection

Filippo Gaudenzi

Supervisor: Prof. C.A. Ardagna

Advisor: Prof. E. Damiani,
Dr. M. Anisetti

Department of Computer Science
"Giovanni Degli Antoni"
Univestitá degli Studi di Milano

This dissertation is submitted for the degree of
Doctor of Philosophy in Computer Science
XXXI Cycle

February 2019

I would like to dedicate this thesis to my beloved parents.

Acknowledgements

I would like to acknowledge all the persons that have helped me during my PhD.

Firstly, I would like to express my sincere gratitude to my advisor Prof. Claudio Agostino Ardagna for the continuous support during my Ph.D study: for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my PhD study.

Besides my advisor, I would like to thank the whole SESAR LAB team, that have been my family for the last 4 years. My sincere thanks goes to Marco, Jonatan, Francesco, Valerio and Fulvio for their insightful comments and encouragement, but also for the hard question which incentivized me to widen my research from various perspectives.

My sincere thanks also goes to Prof. Ernesto Damiani who provided me an opportunity to join his team, without his precious support it would not be possible to conduct this research.

I am also grateful to all the students that helped in carrying out my research with a special thanks to Patrizio and Antongiacomo.

I would like to thank my girlfriend Anthea for putting up with me during this 3 years while I have been facing with the tough PhD way of life.

I end my PhD and my experience at Unimi with the awareness that I don't just leave great colleagues but a wonderful family and awesome friends.

Thank you all.

Abstract

The cloud computing paradigm is changing the design, development, deployment, and provisioning of services and corresponding IT infrastructures. Nowadays, users and companies incrementally rely on on-demand cloud resources to access and deliver services, while IT infrastructures are continuously evolving to address cloud needs and support cloud service delivery. This scenario points to a multi-tenant environment where services are built with strong security and scalability requirements, and cost, performance, security and privacy are key factors enabling cloud adoption.

New business opportunities for providers and customers come at the price of growing concerns about how data and processes are managed and operated once deployed in the cloud. This context, where companies externalise the IT services to third parties, makes the trustworthiness of IT partners and services a prerequisite for its success. Trustworthiness can be expressed and guaranteed through contracts that enforce Service Level Agreements (SLAs), and in a more general way by assurance techniques. By the term security assurance, we mean all the techniques able to assess and evaluate a given target to demonstrate that a security property is satisfied and the target behaves as expected. However, traditional assurance solutions rely on static verification techniques and assume continuous availability of a trusted evaluator. Such conditions are not valid anymore in the cloud that instead requires new approaches that match its dynamic, distributed and heterogeneous nature.

In this thesis, we describe an assurance technique based on certification, towards the definition of a transparent and trusted cloud, from the bare metal to the application layer. The presented assurance approach follows the traditional certification process and extends it by providing continuous, incremental, adaptive and multi-layer verification. We propose a test-based certification scheme assessing non-functional properties of cloud-based services. The scheme is driven by non-functional requirements defined by the certification authority and by a model of the service under certification. We then define an automatic approach to verification of consistency between requirements and models, which is at the basis of the chain of trust supported by the certification scheme. We also present a continuous certificate life cycle management process including both certificate issuing and its adaptation to address contextual changes, versioning and migration.

The proposed certification scheme is however partial if certification of cloud composite services is not supported. Cloud computing paradigm in fact, supports service composition

and re-use at high rates. This clearly affects cloud service evaluation that cannot be simply seen as an assessment on a single target, but it should follow an holistic view that permits to compose certificates. Moreover, while traditional approaches to service composition are driven by the desired functionality and requirements on deployment costs, more recent approaches also focus on SLAs and non-functional requirements. In fact service composition in the cloud introduces new requirements on composition approaches including the need to *i)* select component services on the basis of their non-functional properties, *ii)* continuously adapt to both functional and non-functional changes of the component services, *iii)* depart from the assumption that the cost of the composition is only the sum of the deployment costs of the component services, and also consider the costs of SLA and non-functional requirement verification. In this thesis, we first extended out certification process to evaluate non-functional properties of composite services. We then focus on the definition of an approach to the composition of cloud services driven by certified non-functional properties. We define a cost-evaluation methodology aimed to build a service composition with a set of certified properties that minimizes the total costs experienced by the cloud providers, taking into account both deployment and certification/verification costs.

From the analysis and the definition of certification models and processes, we propose and develop a test-based security certification framework for the cloud, which supports providers and users in the design and development of ready-to-be-certified services/applications. The framework implements a distributed approach to reach all targets at all cloud layers and a paradigm to develop test cases to assess the requested non-functional properties.

The outcome of this thesis is finally validated through an experimental evaluation carried out on real scenarios that *i)*evaluate the assurance of a Web Hosting System provided by the Universitá degli Studi di Milano against the ICT security guidelines for Italian public administration provided by the "Agenzia per l'Italia Digitale" (AgID) and *ii* propose and test a security benchmark for the cloud infrastructure manager OpenStack.

In summary, the contribution of the thesis is manifold: *i)* we design and implement a certification scheme for the cloud, *ii* we extend and adapt the certification of single cloud services to meet cloud composite certification; *iii)* we integrate our certification scheme with the cloud service composition process, developing an algorithm to deploy cloud composite services based on non-functional requirements while minimizing the cost from the cloud service provider point of view; *iv* we design and develop an assurance framework for cloud services certification and validate it in real scenarios.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

The cloud computing paradigm is changing the design, development, deployment, and provisioning of services and corresponding IT infrastructures. Nowadays, users and companies incrementally rely on on-demand cloud resources to access and deliver services, while IT infrastructures are continuously evolving to address cloud needs and support cloud service delivery. This scenario points to a multi-tenant environment where services are built with strong security and scalability requirements, and cost, performance, security and privacy are key factors enabling cloud adoption. New business opportunities for providers and customers come at the price of growing concerns about how data and processes are managed and operated once deployed in the cloud. Traditional assurance solutions rely on static verification techniques and assumed continuous availability of trusted third-parties. Such conditions are not valid anymore in the cloud that instead requires new approaches that match its dynamic, distributed and heterogeneous nature.

1.1 Motivation

Cloud computing paradigm supports a new vision of IT where software and computational resources are released as services over a virtualized ICT infrastructure accessible through the Internet. The convenience introduced by cloud computing in terms of flexibility, and reduced costs of owning, operating, and maintaining the computational infrastructures is radically changing traditional IT provisioning and procurement [1], but it comes at a price of increased risks and concerns. Users deploying a service in the cloud in fact lose full control over their data and applications, which are fully or partially in the hands of cloud providers.

Assurance and verification techniques (e.g., audit, certification, and compliance) need to be adapted to fit the dynamics of the cloud ecosystem [2, 3]. The advent of cloud in fact makes traditional techniques inappropriate, because assurance claims and information are assumed to be all available a priori at the time of evaluation and before service deployment. Cloud assurance aims to increase cloud trust and transparency, and therefore needs to manage

claim verification and evidence collection in a post-deployment environment. Moreover, due to the fact that cloud assurance should manage the complete cloud service/application life cycle, it should *i)* depart from the assumption of a single trusted third party that is available during the whole process and takes responsibility over different claims done on a target object [4], *ii)* implement (semi-)automatic approaches that adapt to changes in the service and/or its environment, *iii)* target multiple cloud layers at the same time.

Effectively tackling such issues is fundamental to increase trust in the cloud [2, 3, 5], and in turn fosters the movement of critical businesses to the cloud. In this thesis, we focus on certification techniques, which aim to implement a secure, trusted, and transparent cloud by specifying a dynamic delegation mechanism supporting multiple signatures of artifacts in a cloud environment.

Recently, the research community has focused on increasing the trust and transparency of cloud and service-based systems by defining cloud-specific security assurance techniques [6–9]. However, while several assurance techniques have been defined for such systems, the market lacks a complete security assurance framework. Among assurance techniques, certification-based assurance has received a lot of attention in the last few years, resulting in the definition of a number of certification schemes [6, 7, 10–12]. Certification schemes provide an independent evaluation process of systems, which results in a signed certificate including a set of claims and the evidence supporting them. Evidence comes from testing, monitoring, and formal proofs. However, certification is a tedious, costly, and time-consuming process, which requires a lot of post-implementation activities, including document writing and code adaptation. Also, existing certification schemes barely address the peculiarities of the cloud, mainly focusing on service layer and providing manual processes. Finally, the lack of supporting certification frameworks makes the deployment of evidence collection tools and techniques at different layers of the cloud stack, as well as the definition of a certification solution that accomplishes cloud events and activities, difficult in practice.

1.2 Contribution of the thesis

The contribution of the thesis is manifold: *i)* the definition of a certification scheme and process for the cloud, *ii)* the definition of a certification method for cloud composite services, *iii)* the definition of a cost-effective deployment algorithm for composite services based on non-functional properties in certificates, *iv)* the development of a cloud service assurance framework implementing the certification schemes at point *i)* and *ii)*. In the remaining of this section we discuss the contribution in more details.

1.2.1 Certification Scheme and Process for the Cloud

Certification has been used to verify software and services for decades, but the nature of the cloud requires a substantial improvement and adaptation: a certification process in the cloud must follow a multi-step process that certifies the support of a given set of non-functional properties by a cloud-based system [12]. The process starts with the certification of the system in a controlled, lab environment. Upon successful verification, the service is deployed in the in-production environment and the certification process is re-executed to prove the support of the same properties verified in the lab environment. Finally, a continuous certification process is set up to monitor the status of the certification process during the system operation. It is based on continuous collection of evidence on the behaviour of the system, which is used to verify whether it maintains the support of the certified properties, according to guidelines designed by the certification authority.

Our scheme supports the above certification process, where the Certification Authority (CA) takes responsibility and signs all evaluation activities to be done to prove a given property, and delegates the management of the certification activities to be done on specific targets to other parties. The correct execution of evaluation activities on real target services can be verified at any time in a semi-automatic way, by checking consistency between requirements provided by the certification authority and activities carried out by the third parties. This check permits to build a chain of trust grounded on certificates between CA, third parties, and final users that increases the trustworthiness of the cloud and its services. The proposed process supports both *i)* basic and traditional certification processes for one-time, static certificate issuing and *ii)* advanced certification processes for continuous, incremental, and multilayer verification.

1.2.2 Certification of Cloud Composite Services

Certification is often seen as a monolithic process where all evaluation activities and evidence belong to the same artefact. Cloud allows users to re-use and compose single cloud service to obtain a new cloud composite service. Security evaluation of the cloud supply chain must consider every service involved in the composition; this however requires anew certification process for each change of the service composition, which is clearly unfeasible in practice. In order to avoid re-certification from scratch and the execution of all evidence collection activities for every new cloud composite service or for any change of component service, it is fundamental to adopt a composite cloud certification that can meet the requirements of this complex scenario.

A certification process for cloud composite services aggregates certificates of component service and their evidence to issue a new certificate for composite service. Moreover, the composite certification should support migration or versioning of the certificates of component services to support in real-time changes of service composition.

1.2.3 Cost-Effective Deployment of Cloud Service Composition

Even if appropriate assurance techniques can foster the adoption of cloud increasing the feeling of trust, there is the need to keep under control the costs observed by cloud providers for certified composition management [13]. These costs can rapidly increase because, in addition to deployment costs considered by existing composition approaches, the costs of continuous certification and verification become not negligible. Current research on cloud computing has privileged solutions minimizing costs on the final users [14–16], neglecting the costs on the cloud providers that often represent the first source of fee increase.

Differently from existing works [14, 15], our service composition approach is driven by certificates awarded to single services and offer a cost-effective solution for the deployment of certified service composition. We provide a fuzzy-based cost evaluation methodology that aims to decrease the costs of cloud providers, also analyzing those costs introduced by the need of keeping the composition continuously monitored and certified.

1.2.4 Assurance Framework for Cloud Services

The lack of appropriate certification frameworks makes the deployment of evidence collection tools and techniques at different layers of the cloud stack, as well as the definition of a certification solution that accomplishes cloud events and activities, difficult in practice. In this thesis we design and develop a test-based security certification framework for cloud-based services that has a twofold value: *i)* it defines a methodology for certification-aware cloud engineering, including a set of guidelines to design and develop certification-ready services/applications; *ii)* it provides a set of modules and components managing the certification process of a generic cloud-based system. The framework can be directly deployed in any cloud stack, supports cloud certification at all layers and manages cloud events affecting the certification itself.

Addressing a more general problem, our framework is complementary to existing solutions for evidence collection (e.g., based on testing [17, 18] and monitoring [19, 20]) and can build on them to implement different certification processes, still maintaining the same deployment strategy. Moreover, our framework integrates a Big Data Platform to properly process all data coming from the cloud. The contribution of our framework is threefold: *i)* it supports online and semi-automatic test-based certification of services/applications; *ii)* it defines an environment and methodology supporting providers in the development of certification-aware services/applications; *iii)* it provides a set of tools supporting the certification-based chain of trust for cloud environments introduced in [21].

To validate the outcomes of this work we carried out a security evaluation by using our framework on two cloud scenarios: *i)* the evaluation of a Web Hosting System provided by the Universitá degli Studi di Milano against the ICT security guidelines for Italian public

administration provided by the "Agenzia per l'Italia Digitale" (AgID) and *ii)*the evaluation of the development of the infrastructure manager OpenStack against a security benchmark.

1.3 Organization of the thesis

This chapter described the motivation and the main objectives of our work outlining the main contributions of this thesis. The remaining of this thesis is organized as follow.

Chapter 2 discusses the state of the art of cloud assurance and security. It provides an overview of the available certification techniques and a comparison between them.

Chapter 3 describes the certification process and models designed and used in this thesis. It also discusses the problem of certificate life-cycle management for cloud services.

Chapter 4 first describes a certification scheme for cloud composite services and then presents an approach to optimize cloud service composition deployment taking into account non-functional properties and cloud service costs.

Chapter 5 presents the framework that implements the proposed certification process. it also introduces Moon Cloud, a business platform for security governance produced as an outcome of the work in this thesis.

Chapter 6 shows two application scenarios where our framework was successfully deployed and executed.

Chapter 7 summarizes the contributions of this thesis and outlines future work.

Appendix A lists all the my personal publications that have been used in this thesis.

Appendix B reports the extensive representation, including XML documents, of two examples in Chapter 3. Moreover, it contains the pseudo-code of the consistency check heuristics we design.

Appendix C contains all the plots of the experimentation presented in Chapter 4.

Appendix D contains a detailed description, including the code, of probes presented in Chapter 6.

Chapter 2

Related works

In the last decade the research community has put a lot of effort on cloud security, since it was identified as one of the main obstacle to cloud adoption [22, 23]. Although cloud security has been substantially improved with new solutions, such as authentication [24, 25] and encryption [26, 27], the trustworthiness of cloud solution still remains a critical problem [28]. Security assurance, which has a wider notion than security, encompasses all the techniques able to assess and evaluate a given target to demonstrate that a security property is satisfied and the target behaves as expected, one of the assurance approach that have been used constantly and successfully in the last 30 years is certification [10].

This chapter provides an overview of the current state of cloud computing assurance and security. The main aim is to give readers the necessary background on cloud security and assurance with a specific focus on cloud certification.

2.1 Cloud Assurance and Certification

Increasing the confidence of cloud is a key factor for cloud adoption. Without the suitable assurance techniques users will be still reticent in fully embrace all the benefits of cloud paradigm. As stated assurance is a wider notion than security, which many sources[1] agreed to define as *"the protection of information and information systems from un-authorized access, use, disclosure, disruption, modification, or destruction"*. Install the best security solutions does not guarantee a secure system; it is necessary to implement a continuous diagnostic process that verifies whether controls are configured in a proper way and behave as expected (security assurance). The term assurance is mostly related to two concepts: quality and reliability. We report in the following some definition of *assurance*:

- *"Information assurance in the field of communication and information systems is defined as the confidence that such systems will protect the information they handle*

[1] See for instance SP 800-37; SP 800-53; SP 800-53A; SP 800-18; SP 800- 60; CNSSI-4009; FIPS 200; FIPS 199; 44 U.S.C., Sec. 3542.

and will function as they need to, when they need to, under the control of legitimate users. Effective information assurance must ensure appropriate levels of confidentiality, integrity, availability, non-repudiation and authenticity." [29]

- *Assurance does not add any additional controls to counter risks related to security, but it does provide confidence that the controls that have been implemented will reduce the anticipated risk. Assurance can also be viewed as the confidence that the safeguards will function as intended." [30]*
- *"A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements" [31]*
- *"Software security assurance: The basis for gaining justifiable confidence that software will consistently exhibit all properties required to ensure that the software, in operation, will continue to operate dependably despite the presence of sponsored (intentional) faults. In practical terms, such software must be able to resist most attacks, tolerate as many as possible of those attacks it cannot resist, and contain the damage and recover to a normal level of operation as soon as possible after any attacks it is unable to resist or tolerate." [32]*
- *"The level of confidence that software is free from vulnerabilities, regardless of whether they are intentionally designed into the software or accidentally inserted later in its life cycle, and that the software functions in the intended manner." [33]*
- *"The level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle and that the software functions in the intended manner." [34]*
- *"The planned and systematic set of activities that ensure that software life cycle processes and products conform to requirements, standards, and procedures." [35]*
- *"Software assurance is a rigorous, lifecycle phase-independent set of activities which ensure completeness, safety, and reliability of software processes and products. This is accomplished by guaranteeing conformance to all requirements, standards, procedures, and regulations" [36]*

Cockling in [37] and IATAC in [32] provide an interesting review on the definition of assurance. For what listed there is a common view on the assurance definition that sometimes even corresponds, as for [33, 34]. These concepts can be mapped on cloud services as follow:

Definition 2.1.1 ***Cloud Service Assurance:*** *All the evaluation activities that should be carried out to establish with a justifiable confidence that a cloud service is acting as expected addressing non-functional requirements.*

Assurance techniques may involve testing, monitoring or formal analysis, auditing or certification activities. Ardagna et al. [38] analyses the state of security and assurance in the cloud. Before focusing on certification and assurance standards, we first provide an overview of assurance techniques for cloud services.

Test. One of the most common approach to evaluate software and services is based on testing [38, 10]. Testing can be carried out both in development and production states and was defined by ISTQB (International Software Testing Qualifications Board) in [39] as *"the process consisting of all lifecycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects."*.

Wu et al. [40] develop a framework for assessing cloud trustworthines through the execution of specific tests which assess performance, data storage integrity and privacy in an automatic way. The works in [17, 18, 41] provide different approaches aiming to verify the behavior of a system at different layers of the cloud, supporting the dynamics and low transparency of the cloud, and increasing the quality of the testing activities. Bai et al. [17] investigate the new architecture and techniques to design testing tools for the cloud and in the cloud providing a literature reviews on some of the available test approaches presented in research papers. Zech et al. [41] focused on the definition of negative requirements derived from risk analysis in security testing. This work proposed a model-driven risk-aware methodology for the security testing of cloud environments. Cloud is not only the target of security testing but it can also used as an enabler to improve and enhance test assurance [38]. An example is Oliveira et al. [42] that present a test framework deployed on the cloud called *CloudTesting* that aims to to parallelize the execution of a test suite over a distributed cloud infrastructure. Gao et al. [18] describe how to design a testing as a service solution that meet cloud issues and challenges.

Monitor. Monitoring assurance can provide a real-time and continuous view of non-functional parameters over a set of targets. While often testing and monitoring may be two sides of the same coin, their meaning and usage are different. Monitor has been widely used in IT systems, from software monitoring (i.e. [43]) to service monitoring (i.e. [44]).

The two main open source monitoring software are Ganglia [45] and Nagios[2] supporting performance monitoring of clusters and grids. Research in the field of cloud monitoring has investigated how to bring the traditional monitoring approaches in the cloud presenting several frameworks [46–48] that tackle the cloud challenge. De Chaves et al. [46] present their experience and issues in building a monitoring system for a private cloud. Alcaraz Alero et al. [47] describe MonPaaS, a monitoring architecture that fits cloud provider and the

[2]https://www.nagios.org/

cloud consumer requirements and that can be integrated with Nagios to monitor OpenStack. Hauser et al. [48] present a monitoring system that collects data only at physical level, this to avoid data duplication and maximize performances. Both Aceto et al. [19] and Alhamazani et al. [49] provide a comparison of monitoring cloud solutions in their surveys.

Audit. Compliance is a fundamental aspect of software and service assurance. Being able to evaluate a cloud service to understand if it is compliant with customer policies or law regulations is a key aspect for cloud assurance. One aspect that research has tried to accomplish is to apply audit techniques to cloud services to make them auditable.

Accountability is a key factor for cloud [50, 51] to take trace of all operations and responsibilities. Agarkhed et al. [52] and Hiremath et al. [53] describe a dedicated framework for data storage auditing in the cloud. Agarkhed et al. [52] provide auditing to check the correctness of outsourced data avoiding that information could be accessed by unauthorized users or hackers in the unsecured cloud network. Hiremath et al. [53] provide a scheme for efficient public auditing technique using Third Party Auditor (TPA) to verify the integrity of data stored in the cloud. Konoor [54] describes how to develop an audit system enable to collect data of the infrastructure manager Openstack and report them using Cloud Auditing Data standard (CADF).[3] Indhumathil et al. [55] present a solution that tries to solve audit in multi-cloud environment by enabling third-party auditors to continuously audit cloud services across different providers. Rasheed et al.[56] present a survey on cloud audit based on user requirements, cloud provider capabilities and techniques for security auditing.

2.1.1 Cloud Certification

Certification implements an assurance process and is often associated to generic quality assurance than security said Spanoudakis et al [10]. Certification has been widely used in the last 30 years, to increase the trust of users in software and services. The International Standard Organization (ISO) defines certification as

"Certification is a procedure by which a third party gives written assurance that a product, process or service is in conformity with certain standards" [57].

Even certification usually comes from an agreement between customers, service supplier and regulatory authorities that often coincide with governments or public association, any private or public body can issue a certificate. In details, ISO defines in [58] the following terms:

- *Authorization: procedure by which an authoritative body gives formal recognition that a body or person is competent to carry out specific tasks, such as certification or evaluation.*

[3] https://www.dmtf.org/standards/cadf

- *Certification Authority: an entity trusted by one or more users to create and assign certificates.*
- *Certificate: declaration by a certification authority confirming that a statement about a software product is valid. Usually issued on the basis of the outcome of an evaluation.*
- *Evaluation: systematic assessment (e.g., by means of tests) of the extent to which a software product is capable of fulfilling a standard.*
- *Evaluation Body: Accredited lab*[4] *carrying out evaluations. The evaluation outcome is a pre-requisite for issuing a certificate on the part of the certification authority.*

Trusted Security Criteria (TCSEC) was the first standard for software security certification in 1985; it was released by the U.S. Department of Defence and was commonly referred as the Orange Book [59]. The TCSEC certification involved a set of tests based on analysis of the software architecture, deployment and user's security requirements; it was a costly and complex certification due the wide area of understanding and analysis. Outside of USA, the first software certification the ITSEC in 1991 in Europe and then the Canadian Trusted Computer Product Evaluation Criteria (CTCPEC) in Canada in the 1993. Nowadays, there are many national and international standards, which are discussed in detail in Section 2.1.2 with reference to cloud system. Before deepen what research has produced on cloud certification, we report the definition of *model-based certification* and *test-based certification* that may by use in largely different practical situation. Damiani et al. [10] define them as follow:

- *"**Model-based certificates** are formal proofs that an abstract model (e.g., a set of logic formulas, or a formal computational model such as a finite state automaton) representing a software system holds a given property. The model to be certified can be provided as a by-product of the software design process, or be reverse-engineered from the software code."*
- ***"Test-based certificates** are evidence-based proofs that a test carried out on the software has given a certain result, which in turn shows (perhaps with a certain level of uncertainty) that a given property holds for that software. In particular, test-based certification of security-related proper- ties is a complex process, identifying a set of high-level security properties and linking them to a suitable set of white- and black-box software tests."*

After software certification, as the technology evolved and changed, reserchers focused on service certification. In the last decade, different approaches to security certification of services have been proposed (e.g., [4, 21, 60–64]). Many of them [4, 61] considered static service certification, with no support for continuous certification of evolving applications.

[4] An accredit lab is lab delegated by a certification authority for evaluation activities.

Common Criteria scheme [65] was the first approach to incremental certification distinguishing between partial re-evaluation and partial re-certification [63]. However, it provides a manual process and puts a high overhead on developers and certification authorities. Anisetti et al. [62] then provided an approach to continuous and incremental certification of SOA and web services, by extending their test-based security certification scheme in [4]. Anisetti et al. [4] propose a model-based testing approach that is capable to automatically generate test-cases for service certification by analysing the service model. Kourtesis et al. [66] provide a method for creating formal method verification of web service through automated tests to test web services. Zhang [64] presents a mobile agent-based tool and an automatic test case generation algorithm to ensure web service trustworthiness. All works were driven by the need of reliability of service-oriented computing infrastructure and lacks of appropriate testing tools and methods, same reasons that are driving research in cloud computing nowadays.

Compared to traditional service certification, cloud certification is: *i) highly dynamic*, it is affected by contextual changes at any layer of the cloud stack, *ii) multi-layer*, it can refer to services at different cloud layers; *iii) intrinsically incremental*, it requires continuous validity verification and incremental adaptation with the scope to minimize costly re-certification activities, and *iv) trustworthy by delegation*, it requires advanced trust models based on delegation to support cloud peculiarities. In this context, Sunyaev and Scheneider [67] discussed the advantages introduced by a certification approach when integrated in a cloud system, while Chen et al. [68] proposed a cloud security assessment indicator system based on classifying and grading. Lins et al. [2] then presented a conceptual architecture for continuous verification of cloud services discussing its benefits and challenges.

Stephanow et al. [69] described a test-based certification framework, based on randomized and non-invasive testing, for evaluating opportunistic providers. This paper, while providing an interesting model, does not focus on continuous certification. Stephanow et al. [70] also presented an approach to continuous certification based on a set of metrics at infrastructure layer. These metrics, which collect evidence of service changes, represent the basis for continuous certification and can be integrated within our certification process (see Chapter 3) to further reduce the need of human intervention. Krotsiani et al. [71] proposed an approach based on monitoring for incremental security certification of cloud services. Their model-based methodology is similar to the one in this thesis, since both works are evolution of the FP7 CUMULUS project. Their paper does not apply to test-based certification, requires full involvement of the certification authority, and provides a high-overhead incremental certification requiring full execution of monitoring activities.

Europe considers cloud certification a key aspect and in these years has put a lot of effort to investigate the subject. Based on the work of the European Union Agency for Network and Information Security (ENISA) [72], European Commision provides a list of supported

certification schemes and standards suitable for the cloud[5]. Europe also contributed to the research on cloud certification by funding several projects:

- *ASSERT4SOA*: ASSERT4SOA aims to produce novel techniques and tools fully integrated within the SOA lifecycle for expressing, assessing and certifying security properties for complex service-oriented applications, composed of distributed software services that may dynamically be selected, assembled and replaced, and running within complex and continuously evolving software ecosystems.[6]

- *CUMULUS*: CUMULUS provides an integrated framework of models, processes and tools supporting the certification of security properties of infrastructure (IaaS), platform (PaaS) and software application layer (SaaS) services in cloud. [7]

- *CIRRUS*: CIRRUS project aims to provide high-level, high impact support and coordination for European ICT security research projects on subject as joint standardization, certification schemes, link research projects with EU policy and strategy, internationalization, as well as industry best practices.[8]

- *SPECS*: Despite it is not focused on certification, SPECS project aims at developing and implementing an open source framework to offer Security-as-a-Service, by relying on the notion of security parameters specified in Service Level Agreements (SLA), and also providing the techniques to systematically manage their life-cycle. [9]

The research community has also focused on different solutions supporting trust in the cloud and implementing different trust models for the cloud [73, 74, 21, 75, 76]. Ryan et al. [73] presented TrustCloud, a framework for accountability and auditability in the cloud, which provides continuous and multi-layer audit of cloud services based on policies and regulations. Although the described trust model and accountability life cycle show some similarities with our approach, they do not provide a formal and rigorous description of how to collect audit-related evidence. Khan et al. [74] discussed the problem of trust in the cloud, underlying the challenges of reduced control and lack of transparency, and acknowledged certification as one of the emerging technologies to bring trust in the cloud. Naskos et al. [75] proposed an approach to deploy and scale cloud services based on security- and performance-related evidence. Wahab et al. [76] proposed a trust-model for cloud services that is completely decentralized. A cloud service trust is built by collecting feedback from its neighbours, which is calculated as the relation between successful interactions over total interactions and spread following a hedonomic coalition game. Differently from the above

[5]https://resilience.enisa.europa.eu/cloud-computing-certification
[6]http://sesar.di.unimi.it/project/assert4soa/
[7]http://sesar.di.unimi.it/project/cumulus/
[8]https://cordis.europa.eu/project/rcn/105735_en.html
[9]http://www.specs-project.eu/project/description/

papers, this thesis describes a trust model based on certification and a running scheme for trusted evidence collection.

Lins et al. [13, 28] investigate the effectiveness of continuous service certification. In [13], Lins et al. analyse if certification can properly increase the user trustworthiness in using cloud services. Through signal theory, they identify which are the keys that most influence its effectiveness. A similar approach has been carried out on analyzing the factor that may drive CSP in continuous service certification adoption. A fundamental aspect identified in this works is certification *cost*.

Cost has been widely treated in research and the main contributions cover cost optimization for users' services deployment [77, 78, 15, 79]. Fittkau et al. [77] propose an approach to minimize cost during VM migration, while Brumec et al. [78] compare the cost between the usage of public cloud and on-premises private infrastructure. Hovewer, for the subject treated in this thesis, studies on CSP costs are more relevant. Greenbergh et al. [80] analyze the cost of cloud from a datacenter perspective listing all the costs needed to run and maintain a cloud datacenter. Li et al. [81] define two cost parameters identified as *i)* total cost of ownership that corresponds to the cost expressed also in [80] and *ii)* utilization cost that is the cost directly associated with the real resources locked up or committed to a particular user or application. An interesting line of research evaluated the costs of service composition [14–16, 82, 79]. He et al. [15] propose three novel QoS-aware service selection approaches for composing multi-tenant service-based systems. Li et al. [79] compare costs and service behaviors from different CSPs, while Medeiros et al. [14] provide different cost patterns which may fit different types of services and service composition. Even though all those papers treat cloud costs and mainly cloud service composition costs, the work proposed in this thesis differs from those since it faces the problem of composition cost by the CSP point of view as a driver to improve cloud certification and cloud trustworthiness.

2.1.2 Cloud Security Standards

Providing the appropriate level of IT governance in a cloud computing environment is a challenging task, indeed understanding the available standards and frameworks that apply to IT governance is a prerequisite to any discussion of governance of or within the cloud. This section reports some of the most used standards: readers that are interested in a more detailed comparison can refer to [83, 84].

ISO 27000-series All the ISO/IEC 27000 family of standards helps organizations keep information assets secure. Among them ISO 27001 is the most known standard for information security management system (ISMS) with more than 30000 certificated companies.[10] BS 7799 is a standard designed and published by the British Standards Institution (BSI) Group

[10]https://www.iso.org/the-iso-survey.htmlaccessedinSeptember2018

ISO	Purpose
ISO/IEC 27000	Information security management systems - Overview and vocabulary
ISO/IEC 27001	Information technology - Security Techniques - Information security management systems - Requirements.
ISO/IEC 27002	Code of practice for information security controls - essentially a detailed catalog of information security controls that might be managed through the ISMS
ISO/IEC 27003	Information security management system implementation guidance
SO/IEC 27004	Information security management - Monitoring, measurement, analysis and evaluation
ISO/IEC 27005	Information security risk management
ISO/IEC 27017	Code of practice for information security controls based on ISO/IEC 27002 for cloud services
ISO/IEC 27018	Code of practice for protection of personally identifiable information (PII) in public clouds acting as PII processors

Table 2.1 ISO table

and composed of several parts that were release between 1995 and 2005. The first part, after a long revision became the ISO 27002, while the second of the BS779, following the same revision process, partially inspired the release of ISO 27001 in 2005. Table 2.1.2 reports some of the ISO 27000 standards.

Even though, Becker et al. [85] state that the actual ISO 27001 may be successfully applied to cloud computing environments, the International Standard Organization released two specific standards for the cloud: ISO 27017, ISO 27018. Both standards target cloud computing but with different purposes:

- ISO 27017 addresses general security of cloud computing providing a set of cloud-specific information security controls supplementing the guidance in ISO/IEC 27002.
- ISO 27018 provides a guidance aimed to ensure that cloud service providers offer suitable information security controls to protect the privacy of their customers' clients by securing PII (Personally Identifiable Information) entrusted to them.

Federal Risk and Authorization Management Program (FedRAMP) FedRAMP provides a standardized approach to security assessment, authorization, and continuous monitoring for cloud products and services used by the US government. Because its goal is to protect US citizens data in the cloud, it is the most rigorous security compliance framework for governments.[11].

The main aim of FedRAMP is to take over all the single requirements and agreements a CSP should face with for each government organization, providing a general and standard

[11]https://www.fedramp.gov/cloud-service-providers/

approach to CSP security approval. The involved actor in the definition and design of FedRamp are: the General Services Administration (GSA), National Institute of Standards and Technology (NIST), il the Department of Homeland Security (DHS), Department of Defense (DOD), National Security Agency (NSA) and the Office of Management and Budget (OMB).

As described in [86], the complex evaluation process of FedRAMP is possible by providing to Executive departments and agencies:

- *Standardized security requirements for the authorization and ongoing cybersecurity of cloud services for selected information system impact levels;*
- *A conformity assessment program capable of producing consistent independent, third-party assessments of security controls implemented by Cloud Service Providers (CSPs)*
- *Authorization packages of cloud services reviewed by a Joint Authorization Board (JAB) consisting of security experts from the Department of Homeland Security (DHS), Department of Defense (DOD), and General Services Administration (GSA);*
- *Standardized contract language to help Executive departments and agencies integrate FedRAMP requirements and best practices into acquisition; and*
- *A repository of authorization packages for cloud services that can be leveraged government-wide.*

FedRAMP release two types of approvals the *i)* readiness and the *ii)* authorization: since today there are 17 ready cloud service providers, 119 authorized and 65 under process of approval.[12] The main distinction between FedRAMP Readiness and FedRAMP Authorized is that the readiness must still undergo an authorization process, while FedRAMP Authorized systems have completed the process at least once.

FedRAMP Ready, carried out by Third Party Assessment Organization (3PAO), asserts that a cloud service is ready for the authorization process and that a Readiness Assessment Report (RAR) has been reviewed and approved by the FedRAMP Program Management Office (PMO). The RAR documents the CSP's capability to meet FedRAMP security requirements. The FedRAMP Ready designation is also required for any cloud service to enter the Joint Authorization Board (JAB) Provisional Authority to Operate (P-ATO) process. FedRAMP Authorized, by comparison, is a designation that is given to systems that have completed the FedRAMP authorization process.

BSI-Compliance Controls Catalogue (C5) In the field of standardisation, any country is free to release its own regulations. Germany, precisely the Federal Office for Information

[12] https://marketplace.fedramp.gov/#/products accessed in September 2018.

Security (Bundesamt für Sicherheit in der Informationstechni - BSI) released the BSI - Compliance Control Catalog (C5) to help cloud customers in having a better overview for a higher level of security and avoiding redundant audits. C5 is organized in 17 sections corresponding to 17 objectives describing the organisational and operational measures and requirements a CSP must accomplish to be compliant. The standard provides basic, additional and optional requirements; of course, in order to be authorized cloud providers must satisfy at least all basic requirements.

The Cloud Security Alliance (CSA) CSA is a not-for-profit organization with the mission to "promote the use of best practices for providing security assurance within Cloud Computing, and to provide education on the uses of Cloud Computing to help secure all other forms of computing". CSA put a significant effort in cloud security and certification. The Cloud Control Matrix (CCM) provides a clear categorisation of security properties and a list of controls and objectives that cover the key areas that are critical to cloud computing. The CCM is a set of roughly 100 controls and assessment guidelines that cover a broad range of security best practices, as well as compliance and regulatory mandates. In fact, the CCM is not only a framework for CSPs, but is also designed to map many compliance standards such as ISO 27001 [122], COBIT [62] or PCI-DSS [186] to avoid duplication of audit activities.

Using the CCM is possible to obtain the CSA Star certification. CSA Star is a rigorous third-party independent assessment of the security of a cloud service provider. The technology-neutral certification is based on the achievement of ISO/IEC 27001 and the specified set of criteria outlined in the Cloud Control Matrix. The assessment is run by an independent third-party authority, and accredited CSA certification body, and for each CCM security domain the CSP is evaluated against five management principles: *i)* communication and stakeholder engagement, *ii)* policies, plans and procedures, *iii)* skills and expertise, *iv)* ownership, leadership, and management, *v)* monitoring and measuring. CSA Star is organized in three levels:

- *CSA Star Level 1: self- assessment.* Answering the CSA Consensus Assessments Initiative Questionnaire (CAIQ), any CSP can assess its own infrastructure.
- *CSA Star Level 2: CSA Star attestation and certification.* The CSA Star level 2 requires CSP to obtain the SOC 2 certification or the ISO 27001 [87].
- *CSA Star Level 3: Continuous Monitoring.* At the moment there is no available certification to determine alignment.

2.2 Cloud Security

Cloud Security is an important problem and a challenging issue as the number of papers and surveys on the topic confirm [38, 88–94]. Cloud Security Alliance's 2018 report [95] reported

as the 12 biggest threats to cloud computing, ranked in order of severity per survey results, the following:

- Data Breaches
- Insufficient Identity, Credential and Access Management
- Insecure Interfaces and APIs
- System Vulnerabilities
- Account Hijacking
- Malicious Insiders
- Advanced Persistent Threats (APTs)
- Data Loss
- Insufficient Due Diligence
- Abuse and Nefarious Use of Cloud Services
- Denial of Service
- Shared Technology Vulnerabilities

In the Section we analyse some of those threats and report related works.

Data breaches are often the goal of targeted attacks, but it might occur that they are the result of human errors, application vulnerabilities or just poor security policies. Data in the cloud must be protected, and so the privacy, and the level of security must be appropriate with value of data we are managing (e.g. health information, personally identifiable information (PII). Research has investigated confidentiality, privacy and storage security widely. Barona et al. [96] present a survey discussing reasons and consequences of data breaches. Kolevski et al. [97] review literature in cloud computing data breaches by using a socio-technical approach which encapsulates the three major dimensions: social, technical, and environmental. Kirkman [98] proposes a solution to track and move data securely on the cloud by using smart contract and blockchain technology. Jajodia et al. [99] consider the problem of how to securely backup encryption keys for increasing data safety and availability, reducing the risk of data loss due to unavailability of keys, and limiting the risk of key disclosure and confidentiality breach. The authors present a scheme called recoverable encryption through a noised secret that permits to store key backups on a single machine, and is robust to decryption by brute force attacks.

Lack of scalable and secure identity access management systems are one of the main reasons of cyberattacks and data breaches in the cloud. The adoption of multi-factor authentication,

which uses multiple authentication factors to grant access, may increase the security of cloud services [95]. Fathi et al. [100] propose a progressive multi-factor authentication that uses a combination of explicit and implicit factors that aims to minimize the perceived authentication hardship in using it. Choudhury et al. [101] propose a framework for identity management, mutual authentication, session key establishment designed for the cloud.

The report in [95] states that programming interfaces (APIs), as the most exposed part of a system, should be designed to be resilient against both accidental and malicious attempts to circumvent policy. Bahaweres et al. [102] show how private cloud without an appropriate security system can be easily vulnerable to Distributed Denial of Service (DDoS). Gracia-Tinedo et al. [103] show how freemium account could be abused by malicious attacker; in fact, by automating freemium account creation and usage, it is possible to run attacks such as DDoS.

Cloud Computing provides a sharing environment where applications from various organizations may reside in proximity one to each other sharing computational resources such as memory and CPU. Razavi et al. [104] present a co-resident VM to get access to the victim VM; Hasan et al. [105] propose a mitigation defense by applying game theory to the problem, while Liang et al. [106] propose a different solution to the same problem, by optimizing the placement of the VMs at deployment time following a VM placement strategy.

A malicious insider is a *"current or former employee, contractor, or other business partner that, leveraging her authorized access to network, system, or data, intentionally misuses that access in a manner that negatively affect the confidentiality, integrity, or availability of the organization's information system"* [107]. Nkosi et al. [108] present a detecting system for cloud environment aiming to detect malicious insiders by monitoring useres behaviour patterns. Rocha et al. [109] present a work which demonstrates how is possible for a malicious insider to access data by using a crafted attack in a Xen-powered cloud infrastructures. The authors also present a mitigation solution by implementing a lightweight mandatory memory access control mechanism for Xen.

In summary, accountability, confidentiality, integrity and authorization are key properties to avoid any of the threats mentioned by CSA in [95]. Ardagna et al. [38] survey hundreds of articles, organizing them over those properties. Rukavitsyn et al. [110] present an approach where cloud users may use separate services for authentication, data management and meta data storage to limit the possibility of data loss and corruption. Sevis et al. [111] analyse the state of the art of data integrity in the cloud comparing different techniques such as provable data possession and proof of retrievability.

2.3 Contribution to the State of the Art

The last 30 years were characterized by an important evolution of assurance techniques [10]. This chapter presented an excursus on assurance and security approaches starting from the

Orange Book [59] and Common Criteria [65] and ending with the last proposal from the research communities such as [55, 53, 71].

Mainly focus on certification activities, the approaches were still mostly based on one time certification, due to the complexity of certification process that involves actively several actors and poses the problem of the substantial costs to be borne [13]. In the context of cloud, which is highly dynamic and intrinsically incremental and involves contextual changes at any layer of the cloud stack, one time certification is obsolete and outdated. Common Criteria proposed the first approach to incremental certification distinguishing between partial re-evaluation and partial re-certification [63], but requiring a complex manual process. The work in this thesis embraces the concepts of continuous certification presented in [62, 2] but extending them providing a certification process which supports incremental, and multilayer verification.

Multilayer verification approaches are presented in literature [73, 74] but they provide specific solutions for specific purposes. The framework presented in this thesis aims to be a general purpose framework for cloud service assurance evaluation, this means that may be used to apply several certification schemes such as CSA or C-5. Moreover these standards mostly described general requirements that must be turned then into specific audit activities. Chapter 3 shows how the adoption of the certification model template and instance allows CA to specify generic requirements on the template and Accredit Lab to build a specific instances based on the given template. Moreover, to minimize manual operation we provide an automatic matching algorithm that can validate an instance over a template.

Many of the certification frameworks available on the market and proposed by the research community lack a formal representation of the certification process suitable for the cloud. Moreover, the work presented in this thesis combines both description of the activities and the effectiveness of collecting evidence to prove a given security property. Assessments, such as CSA-Star [61], mainly rely on survey and self-assessment, while the framework presented in this thesis collects evidence of the behavior of the service under certification.

All the aspects considered in this thesis have always been correlated to certification costs. Lins et al. [13] and before Common Criteria [63] highlighted how certification is a costly operation and how important this factor is for its adoption. Certification process automation, which tries to avoid the involvement of CA, aims to take the costs under control, as well as the composition and re-use of available evidence and models to build new certifications (see Chapter 4). This thesis faces the problem of cost optimization from the point of view of the Cloud Service Provider with the aim to enhance the fruition of security cloud services.

Figure 2.1 shows the main contributions that this thesis brings to the state of the art and that can be summarized as follow: *continuous, incremental and multi-layer* certification with high degree of *automation*, all represented and defined in a *formal* and *declarative* form. In more details Figure 2.1 describe the evaluation and characteristics

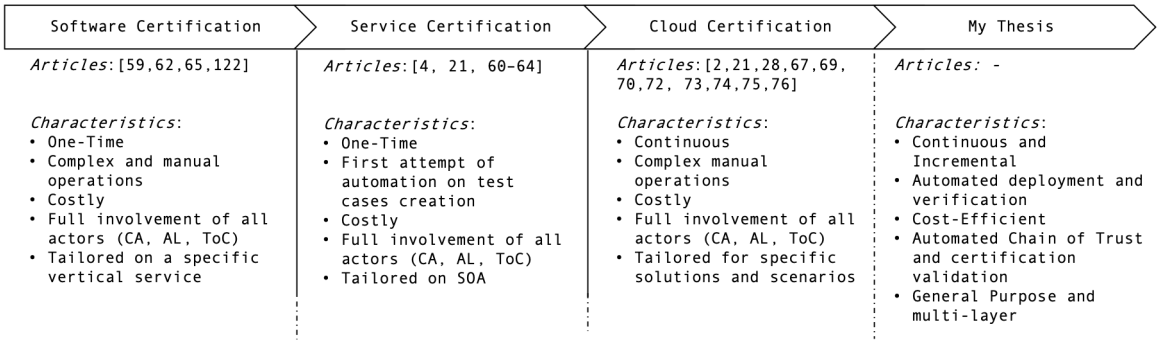


Fig. 2.1 Evaluation of certification and comparison with thesis contribution.

As a breakpoint, the work in this thesis was the corner stone to build a business product "Moon Cloud", which aims to evaluate security of cloud services.

2.4 Chapter Summary

Cloud has already changed the provisioning of IT resources, but security is still the main concern and obstacle to its adoption. Research has been working hard to improve security features, and meet user and technology requirements. The effectiveness of cloud security can be measured by the number of attacks and threats, but the main issue in cloud adoption is the feeling of security perceived by final user. Assurance techniques (test, monitoring, certification and audit) may have the chance to increase the cloud trustworthiness and transparency. This chapter reviewed research on cloud security providing the state of the art of the main cloud security threats and cloud assurance techniques.

Chapter 3

Cloud Service Certification: Process and Models

Cloud computing paradigm supports a new vision of IT where software and computational resources are released as services over a virtualized ICT infrastructure accessible through the Internet. The convenience introduced by cloud computing in terms of flexibility, and reduced costs of owning, operating, and maintaining the computational infrastructures, comes at a price of increased risks and concerns. Users deploying a service in the cloud in fact lose full control over their data and applications, which are fully or partially in the hands of cloud providers.

Assurance and verification techniques (e.g., audit, certification, and compliance) need to be adapted to fit the dynamics of the cloud ecosystem [2, 3]. The advent of cloud in fact makes traditional techniques inappropriate, because assurance claims and information are assumed to be all available a priori at the time of evaluation and before service deployment. Cloud assurance aims to increase cloud trust and transparency, and therefore needs to manage claim verification and evidence collection in a post-deployment environment. Moreover, due to the fact that cloud assurance should manage the complete cloud service/application life cycle, it should *i)* depart from the assumption of a single trusted third party that is available during the whole process and takes responsibility over different claims done on a target object [4], *ii)* implement (semi-)automatic approaches that adapt to changes in the service and/or its environment, *iii)* target multiple cloud layers at the same time.

Effectively tackling such issues is fundamental to increase trust in the cloud [2, 3, 5], and in turn fosters the movement of critical businesses to the cloud. This thesis focuses on certification techniques, which aim to implement a secure, trusted, and transparent cloud by specifying a dynamic delegation mechanism supporting multiple signatures of artifacts in a cloud environment.

A certification process in the cloud must follow a multi-step process that certifies the support of a given set of non-functional properties by a cloud-based system. The process

starts with the certification of the system in a controlled, lab environment. Upon successful verification, the service is deployed in the in-production environment and the certification process is re-executed to prove the support of the same properties verified in the lab environment. Finally, a continuous certification process is set up to monitor the status of the certification process during the system operation. It is based on continuous collection of evidence on the behavior of the system, which is used to verify whether it maintains the support of the certified properties, according to guidelines designed by the certification authority.

This Chapter [1] presents a certification scheme that supports the above certification process, where the certification authority takes responsibility and signs all evaluation activities to be done to prove a given property (requirements specified in a *certification model template*), and delegates the management of the certification activities to be done on specific targets to other parties (activities specified in a *certification model instance*). The correct execution of eva luation activities on real target services can be verified at any time in a semi-automatic way, by checking consistency between certification model template and instance. This check permits to build a chain of trust grounded on the requirements specified by a certification authority in a certification model template, increasing the trustworthiness of the cloud and its services. The proposed process supports both *i)* basic and traditional certification processes for one-time, static certificate issuing and *ii)* advanced certification processes for continuous, incremental, and multilayer verification.

3.1 Basic Concepts

In this section we present the involved actors and requirements characterizing our certification process, which will be used in the remaining of this thesis.

3.1.1 Actors

A certification scheme for the cloud implements a continuous process whose goal is to verify whether a cloud service holds a given (set of) property. The cloud service under evaluation is referred to as *Target of Certification* (*ToC*), while property p is composed of a controlled name $\hat{p}$ (e.g., confidentiality of data in transit) and a level l modeling the strength of the supported property, as defined by the Cloud Security Alliance [112].

A certification process for the cloud is a collaborative effort involving *i)* a *service provider* developing cloud-based applications that need to be certified (TOC); *ii)* a *cloud provider*

[1] This chapter is based on the following publications:

- A semi-automatic and trustworthy scheme for continuous cloud service certification, Co-author: M. Anisetti, C.A. Ardagna, E. Damiani, published in IEEE Transactions on Services Computing, 2016
- Modeling time, probability, and configuration constraints for continuous cloud service certification, Co-author: M. Anisetti, C.A. Ardagna, E. Damiani, N. El Ioini , published in Computers & Security (COSE) 72, 2018

(CSP) that either supports application certification or wants to certify its own cloud services;[2] *iii)* a *certification authority (CA)* responsible for the design and definition of certification requirements and methodology; *iv)* an *accredited lab (AL)* delegated by the certification authority and responsible for certification activities.[3] This structure reflects the traditional certification process (Section 2.1.1) .

Our certification process is driven by a Certification Authority that manages all certification activities leading to certificate issuing. It is composed of two sub-processes: *i) evidence collection sub-process* and *ii) life cycle sub-process*. The *evidence collection sub-process* collects the evidence at the basis of a trustworthy certification and is carried out by the certification infrastructure. The *life cycle sub-process* implements a continuous certification process that accomplishes the evolution of the *ToC*, managing *ToC* migrations and versioning. The whole process is described in more details in Section 3.3.

3.1.2 Requirements

A cloud certification process should satisfy new requirements to fit the dynamic, distributed, heterogeneous and high rate changing peculiarities of cloud environments.

- *Generic and multilayer.* The certification process must be generic to cover not only cloud services, but any service. This permits to use the same certification process among all the cloud stacks and to keep the compatibility with older approaches.
- *Continuous and Holistic.* The evaluation process should be continuous to support the dynamics of the environment and should indeed become holistic. Since non-functionals requirements and deployment configurations change over time, the certification process should depart from a one-shot activity executed in a controlled environment.
- *Based on Evidence.* The certification issuing process must be based on evidence produced by evaluation activities that can proof a given property. The evidence validates the property and moreover increases the certification trustworthiness.
- *Modular.* Certification process should be designed to support certification of composite services based on activities, evidence and outcomes of the certification of the single component services.
- *Verifiable and Repeatable.* Once defined and described, a certification process must be verifiable in terms of required activities to issue a property and repeatable in time in any conditions.

[2] Often service provider may coincide with cloud service provider.
[3] An accredited lab is an official emanation of a certification authority carrying out a system evaluation.

- *Support for migration and versioning.* Being the cloud dynamic and heterogeneous, the certification process should support single property overwrite (migration) and versioning of a property within a single service (more details in Section 3.5.2 and Section 4.2).
- *Support trust by delegation.* The certification process must provide a chain of trust where the delegation mechanism can be executed, when possible, in an automatic manner without the countinous involvement of the certification authority. In this context provide an autonomous approach where the certification process can be expressed through machine-readable documents can lead to an automatic validation of the certification process itself.

3.1.3 Terminology

Chapter 3 discusses all the components of the certification process and models at the basis of this thesis. Table 3.1 summarizes all the acronyms used in this chapter with the aim to help readers in keeping trace of the many terms.

$\mathcal{C}$	Certificate
$\mathcal{I}$	Certification Model Instance
$\mathcal{T}$	Certification Model Template
p	Non-Functional Properties
$\hat{p}$	Non-Functional Properties name
A_p	Set of property attributes
θ	Non-Functional Mechanisms
$\hat{\theta}$	Non-Functional Mechanisms name
A_m	Set of of attributes specifying mechanism configuration
ToC	Target of Certification
Θ	set of mechanisms
b	layer (service, platform, infrastructure,..)
ev	Evidence
tc	Test Case
Pr	Preconditions
In	Test Inputs
EO	Expected Output
Po	Postconditions
l	Life Cycle
$f^{\triangleright}$	CM Instance validity check

Table 3.1 Acronymous Table

3.2 Certification Building Blocks

We describe the building blocks of our certification scheme for the cloud.

Non-Functional Properties (p). A non-functional property p is a pair $(\hat{p},A_p)$, where $\hat{p}$ is an abstract property and A_p is a set of attributes refining it. An abstract property is taken from a shared vocabulary (e.g., confidentiality, integrity, availability) [4] or domain-specific vocabularies derived from regulations (e.g., [113]), standards (e.g., [87]), cloud security specifications (e.g., [61]). Attributes can include information on the class of mechanisms guaranteeing $\hat{p}$ (e.g., access control, encryption, signature), a level modeling property strength (e.g., CVSS standard severity score level *low*, *medium*, *high*), and contextual attributes (e.g., confidentiality of data *in transit*, availability with *95% uptime*). Differently from existing work (e.g., [4, 61]), we decouple definition of properties from the mechanisms that must be implemented to support them. Non-functional properties are organized in a hierarchy . Given two properties $p_i,p_j\in\mathcal{P}$, p_i is weaker than p_j (denoted $p_i\preceq_P p_j$) if $p_i.\hat{p}=p_j.\hat{p}$, and $\forall a_k\in A_p$, either the value $n_i(a_k)$ of attribute a_k is not specified or $n_i(a_k)\preceq_{a_k} n_j(a_k)$. We note that total order relations $\preceq_{a_k}$ between contextual attributes $a_k\in A_p$ can be defined by expert users. An example of property hierarchy can be found in [4].

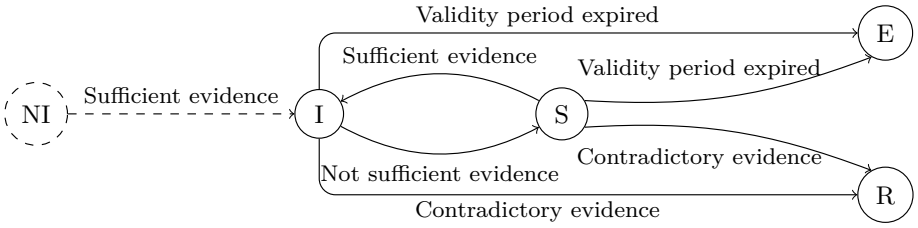
Non-Functional Mechanisms (θ). A non-functional mechanism θ is a pair $(\hat{\theta},A_m)$, where $\hat{\theta}$ is a mechanism type (e.g., access control, encryption), and A_m is a set of attributes specifying mechanism configurations (e.g., cloud layer, encryption algorithm, key length) and specific cloud stack configurations affecting the mechanism behavior. Non-functional mechanisms support the verification of properties and are organized according to their type $\hat{\theta}$. Abstractions can be defined over mechanisms, possibly introducing a hierarchy $\mathcal{H}_{M_i}$ for each type $\hat{\theta}_i$. A hierarchy $\mathcal{H}_{M_i}$ is then defined as a pair $(\mathcal{M}_i,\preceq_{M_i})$, where $\mathcal{M}_i$ is the set of all mechanisms of a given type, and $\preceq_{M_i}$ is a partial order relationship over $\mathcal{M}_i$. The partial order is defined by domain experts (e.g., a certification authority – CA) so that given two mechanisms $\theta_j,\theta_k\in\mathcal{M}_i$, θ_j is weaker than θ_k (denoted $\theta_j\preceq_{M_i}\theta_k$) if $\theta_j.\hat{\theta}=\theta_k.\hat{\theta}$, and $\forall a_t\in A_m$, either $n_j(a_t)$ is not specified or $n_j(a_t)\preceq_{a_t} n_k(a_t)$. We note that total order relations $\preceq_{a_t}$ between mechanism attributes $a_t\in A_m$ can be defined by expert users. We also note that there is a special hierarchy $\mathcal{H}_{M_f}=(\mathcal{M}_f,=)$, where $\mathcal{M}_f=\{(\textit{Functional},\{\})\}$, referring to all mechanisms concerning functional aspects of a given service. All families can be logically seen as part of a common hierarchy $\mathcal{H}_M$ of mechanisms $\mathcal{M}$ having a common ancestor denoted as *any*. Each mechanism is annotated with a set $\{\textit{events}\}$ of events affecting its execution and, in turn, the validity of an existing certification process. Events can also refer to specific cloud configurations, which are requested for the correct functioning of the mechanism.

Target of Certification (ToC). The meaning of properties is strictly associated with the application context and the cloud perimeter, called *Target of Certification* (ToC), they insist on. ToC describes the mechanisms, possibly at different cloud layers, behind a non-functional property. ToC is defined as (Θ,b), where $\Theta=\{\theta_i\}$ is a set of mechanisms $\theta_i\in\mathcal{M}$ and b specifies the layer (i.e., service, platform, infrastructure) of certificate binding. Each

mechanism belongs to a cloud layer and can support a property alone or in cooperation with other mechanisms in *ToC*. The certificate, instead, is bound to a single layer b representing the provisioning layer for the certified service. For instance, let us consider a *ToC* for security property p=(*Confidentiality*,{`ctx`=*in-transit*/*at-rest*}). *ToC* includes two mechanisms θ_1 and θ_2 deployed at service layer and infrastructure layer, respectively, and its binding is defined at service layer (i.e., b=<*service*>). Mechanism θ_1=(*encryption*,{`algo`=*XML-encryption*,`protocol`=*WS-Security*,`level`=*message-in-transit*}) refers to a mechanism implementing an encrypted communication channel, mechanism θ_2=(*encryption*,{`algo`=*encrypted FS*}) identifies a mechanism implementing an encrypted file system.

Evidence *ev* A certification process (see Section 3.3) relies on the collection of evidence at the basis of a certificate, proving a non-functional property p for a given *ToC*. We focus on test-based evidence *ev* that includes *i)* the specification of the collection process, *ii)* the set of testing activities (i.e., test cases) to be executed, *iii)* the results retrieved by test case execution and corresponding rules for their aggregation, and *iv)* a reference to the mechanisms specified in the *ToC* over which test cases are executed and corresponding results collected. For simplicity, but without loss of generality, test-based evidence is defined as ev={{(θ,*Pr*,*In*,*EO*,*Po*)}}, where {(θ,*Pr*,*In*,*EO*,*Po*)} represents a single test case *tc* as a sequence of 5-tuples (θ,*Pr*,*In*,*EO*,*Po*), with θ a mechanism, *In* the set $\{i_1,\ldots,i_n\}$ of inputs, *EO* the set $\{eo_1,\ldots,eo_m\}$ of (expected) outputs, *Pr* the set of pre-conditions, *Po* the set of post-conditions. Pre-conditions and post-conditions express dependencies between inputs and (expected) outputs. Evidence collection follows a model-based testing approach and is driven by an automaton describing all activities, in the form of a sequence of invocations, to be done on the *ToC*. The automaton, called evidence collection model m, is described as a Symbolic Transition System (STS) [4] and combines the automaton m_θ of the mechanisms $\theta \in \Theta$ in *ToC*. Model m is then defined as $\langle \mathcal{S}, s_0, \mathcal{V}, \mathcal{I}, \mathcal{A}, \rightarrow \rangle$, where $\mathcal{S}$ is a set of states s, each one referring to either a mechanism θ defined in *ToC* or a functional mechanism, $s_0 \in \mathcal{S}$ is the initial state that refers to a functional mechanism modeling *no operation*, $\mathcal{V}$ is the set of internal variables, $\mathcal{I}$ is the set of interaction variables, $\mathcal{A}$ is the set of actions (i.e., service operations and internal function calls), and $\rightarrow$ is the transition relation. $\rightarrow$ consists of a set of edges connecting two states and labeled with an action, a guard in disjunctive normal form (conditions on transition), and an update mapping (new assignments to variables). According to m and its linear independent paths modeling the testing flows $\phi_i \in \Phi(m)$, evidence *ev* can be generated following our approach in [4]. Each test case *tc* refers and exercises a given flow ϕ_i.

Life Cycle, (l). The certificate life cycle l models the certificate evolution from its issuing to possible expiration or revocation. In traditional certification, it is in the bailiwick of the *CA* issuing the certificate. Decisions like certification issuing, suspension, revocation, or



Conditions on transitions

$cond_{e_{NI,I}}$=0.9tc $cond_{e_{S,R}}$=$\neg cond_{e_{NI,I}} \wedge$ *suspension_time*>t

$cond_{e_{I,E}}$=*expiration date is reached* $cond_{e_{S,I}}$=$cond_{e_{NI,I}}$ $cond_{e_{I,R}}$=$\emptyset$

$cond_{e_{I,S}}$=$\neg cond_{e_{NI,I}} \wedge$ *suspension_time*<t $cond_{e_{S,E}}$=*expiration date is reached*

Fig. 3.1 Life cycle with states Not Issued (NI), Issued (I), Suspended (S), Expired (E), Revoked (R) and examples of conditions on transitions.

expiration are normally taken asynchronously, statically, and offline by the *CA*, for instance, as a reaction to new discovered vulnerabilities or audit activities. In a cloud scenario, where the certificate life cycle is managed at run-time on the basis of evolving evidence, the static intervention of a *CA* is not always feasible. The life cycle is modeled as a deterministic finite state automaton $G^l(V^l,E^l)$ with each vertex $v \in V^l$ representing a possible state of the certificate with label $label(v)$ (e.g., issued, suspended) and each edge $e=(v_i,v_j) \in E^l$ representing a transition between two states. Each edge e is labeled with a condition $cond_e$ over certificate evidence that regulates the transition. Figure 3.1 shows an example of the life cycle automaton with transition conditions. For instance, edge $e_{NI,I}$ is labeled with a condition $cond_{e_{NI,I}}$ requiring that at least 90% of the test cases are successful (i.e., $cond_{e_{NI,I}}$=0.9tc) for the certificate to move from state NI to state I.

3.3 Certification Models and Process

Our certification process is responsible for all evaluation activities aimed to produce a certificate for the *ToC* (*issuing phase*), as well as to continuously verify the validity of the certificate against context changes to reduce unnecessary certificate revocation and re-certification [114] (*post-issuing phase*). It is based on specific machine-readable documents, namely Certification Model (CM) Template, Certification Model (CM) Instance, and Certificate.

3.3.1 Certification Model Template

CM Template $\mathcal{T}$ drives the definition of the certification methodology, and represents the cornerstone for building a chain of trust that is grounded on the correctness of the certification methodology itself. It is an abstract representation of the process inputs, which specifies high-level requirements, configurations, and activities for the certification of a property for a given (class of) *ToC*. It is defined as 5-tuple of the form $\langle p,ToC,m,ev,l \rangle$ and signed by

the certification authority. CM Template $\mathcal{T}$ specifies requirements on *ToC* in terms of the mechanisms to be implemented to support a property p, the model m for evidence collection, and the evidence ev to be collected. It also includes a life cycle l for continuous verification of certificate validity. Mechanisms $\theta \in \Theta$ in *ToC* can be defined at different levels of abstraction, from mechanisms only including the type (i.e., $(\hat{\theta},\emptyset)$) to fully specified ones (i.e., $(\hat{\theta},A_m)$). The evidence collection model m describes execution flows $\Phi(m)$ involving mechanism in Θ, which must be evaluated to certify p. Evidence ev must be produced according to $\Phi(m)$. We note that evidence ev in $\mathcal{T}$ specifies test cases $\{(\theta,Pr,In,EO,Po)\}$, where inputs In and expected outputs EO are expressed in the form of partitions of the corresponding input and output domains, $\mathcal{D}_{In}$ and $\mathcal{D}_{EO}$, respectively. A partition of a domain $\mathcal{D}$ is a set $\phi(\mathcal{D})=\{D_1,\ldots,D_n\}$ of equivalence classes such that: $\bigcup_{j=1}^{n} D_j=\mathcal{D}$ and $\forall j\neq t\ D_j\cap D_t=\emptyset$ with $j,t=1,\ldots,n$. For instance, equivalence classes $\phi(\mathcal{D}_{pwd})$ for input parameter *password* can include: *i)* the set of valid passwords, *ii)* the set of invalid passwords, *iii)* the set of valid but not existing password.

Example 3.3.1 ($\mathcal{T}$) *Let us consider the cloud storage service for a payment system deployed on top of Openstack, to be certified for property* Confidentiality*. A CM Template $\mathcal{T}$ can include security property p=(*Confidentiality*, {*`ctx`=in-transit/at-rest*}) and target of certification ToC=({θ_1, θ_2, θ_3}, <*service*>), where θ_1=(*encryption*, {*`level`=message-in-transit*}) is an encryption mechanism securing public communication channels, θ_2=(*encryption*, {*`level`=internal-communications*}) is an encryption mechanism securing internal cloud service communications, θ_3=(*encryption*, {*`level`=data-at-rest*}) is an encryption mechanism securing stored data. Evidence collection model m in $\mathcal{T}$ specifies the flows of execution, which are used for test case generation, by combining mechanisms in ToC. It generates a set of test cases ev expressed in terms of test partitions $\mathcal{D}_{In}$ and $\mathcal{D}_{EO}$ insisting on the above mechanisms. The life cycle automaton is fully defined in terms of states that can be assumed by a certificate with all mandatory transition conditions specified in terms of specific aggregations on evidence.*

The full XML representation of Example 3.3.1 can be found in Appendix B.

3.3.2 Certification Model Instance

A Certification Model Instance ($\mathcal{I}$) is a procedural, executable model generated by refining and instantiating $\mathcal{T}$ on a real *ToC*. CM Instance $\mathcal{I}$ includes specific information on configurations and activities to be executed on the service under evaluation for evidence collection. It is jointly specified by the accredited lab and the service provider, and can be defined as a 5-tuple of the form $\langle\overline{p},\overline{ToC},\overline{m},\overline{ev},\overline{l}\rangle$. We note that CM Instance $\mathcal{I}$ can be not unique for CM Template $\mathcal{T}$.

CM Instance $\mathcal{I}$ is under the responsibility of the accredited lab and contributes to the establishment of a complete chain of trust, which is grounded on the corresponding

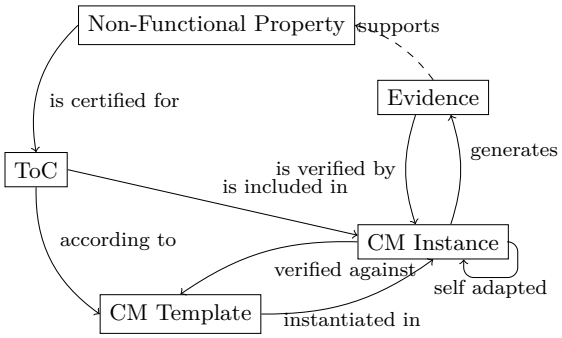


Fig. 3.2 Conceptual framework

CM Template signed by the certification authority (see Section 3.6). Before distributing a CM Instance, the accredited lab verifies *i)* the signature of the CM Template from which it is generated/to which is claiming conformance, *ii)* that the CM Instance is a correct instantiation of the CM Template and correctly represents the ToC (see Section 3.2). The instantiated target of certification $\overline{ToC}$ contains all configurations for the mechanisms in the real service implementation. Evidence $\overline{ev}$ defines test cases specifying the reference to the real mechanisms θ, input values i, and expected output values eo. We note that an input value i (expected output value eo, resp.) can be considered as a representative of the corresponding equivalence class D_j *iff* $i \in D_j$ ($eo \in D_j$, resp.). In the following, we denote an equivalence class D_j as $[i]$, where $i \in D_j$. Additional information (e.g., access credentials, cloud configurations, network configurations, endpoints) regarding the mechanism implementation is fundamental for evidence generation.

Example 3.3.2 ($\mathcal{I}$) *Let us consider CM Template $\mathcal{T}$ in Example 3.3.1. A CM Instance $\mathcal{I}$ for $\mathcal{T}$ can include the following elements. Security property* $\overline{p}$=(Confidentiality, {`ctx`=in-transit/at-rest}). *Target of certification* $\overline{ToC}$=({θ_1, θ_2, θ_3}, <service>), *where* θ_1=(encryption, {`algo`=XML-encryption, `protocol`=WS-Security, `level`=message-in-transit}) *implements a XML-encryption mechanism based on WS-Security securing public communication channels,* θ_2=(encryption, {`level`=internal-communications, `algo`=HTTPS}) *implements an HTTPS communication channel securing internal cloud service communications,* θ_3=(encryption, {`level`=at rest, `algo`=encrypted FS}) *implements an encrypted file system securing stored data. The remaining components should be such that:* i) $\overline{m}$ *is consistent with* m *in* $\mathcal{T}$, ii) $\overline{ev}$ *includes test cases reflecting the partitions in* $\mathcal{T}$, iii) $\overline{l}=l$.

We introduce an instantiation function $\xrightarrow{I}$ that produces $\mathcal{I}$ as specialization of a given $\mathcal{T}$ as follows.

Definition 3.3.1 ($\xrightarrow{I}$) *Let* $\mathcal{T}=\langle p, ToC, m, ev, l\rangle$ *be a CM Template. An instantiation* $\xrightarrow{I}$ *is a transformation that takes* $\mathcal{T}$ *as input and produces* $\mathcal{I}=\langle \overline{p}, \overline{ToC}, \overline{m}, \overline{ev}, \overline{l}\rangle$ *as output where:* $p \xrightarrow{I} \overline{p}$, $ToC \xrightarrow{I} \overline{ToC}$ $m \xrightarrow{I} \overline{m}$, $ev \xrightarrow{I} \overline{ev}$, $l \xrightarrow{I} \overline{l}$.

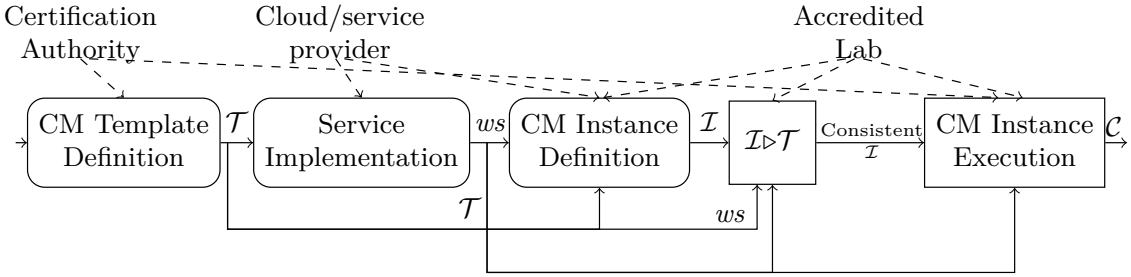


Fig. 3.3 Certification process: squared boxes represent steps that can be completely automatized, rounded boxes steps for which we provide tools or guidelines for cloud providers, accredited labs, or CA.

3.3.3 Certificate

CM Instance $\mathcal{I}$ describes an executable evidence collection process (see Figure 3.2) whose results are evaluated for: *i) certificate issuing*, if the produced evidence is sufficient according to the certification authority, *ii) certificate adaptation*, to continuously validate and possibly adapt the status of a certificate $\mathcal{C}$, following certificate life cycle $\bar{l}$ (see Section 3.2). A certificate $\mathcal{C}$ is a 4-tuple $\langle \mathcal{I}, \mathcal{T}, ws, \overline{ev}_r \rangle$, where $\mathcal{I}$ is a CM Instance, $\mathcal{T}$ the original methodology over which $\mathcal{I}$ is defined, ws the certified service, and $\overline{ev}_r$ the results of the execution of test cases in $\overline{ev} \in \mathcal{I}$. Certificate $\mathcal{C}$ plays a fundamental role for improving the trustworthiness of cloud services/systems and is at the basis of advanced processes such as certification-aware service discovery and service composition, to name but a few.

3.3.4 Certification Process

Figure 3.2 shows the conceptual framework at the basis of our certification process, as an extension of the one in [21]. The certification process aims to prove a property for a *ToC* and is driven by a CM Instance. CM Instance is generated as a refinement of a given CM Template and verified against it through the validity check in Section 3.4. The evidence supporting the considered property is continuously generated, according to the CM Instance. We remark that, in some cases, the evidence is not sufficient to prove a given non-functional property (dashed arrow in Figure 3.2) and therefore award the corresponding certificate. Based on the conceptual framework, Figure 3.3 presents our certification process that is composed of five main steps. In the first step (*CM Template Definition*), the certification authority produces a CM Template $\mathcal{T}$ specifying the certification methodology for proving p on a class of *ToC*. We note that *CM Template Definition* might happen well before the execution of the other steps of the certification process, making certification authority involvement feasible also in a cloud environment. In the second step (*Service Implementation*), the cloud/service providers implement the service ws to be certified. In the third step (*CM Instance Definition*), the

accredited lab, with the help of the cloud/service providers, produces CM Instance $\mathcal{I}$ for the implemented service as a refinement of the methodology in the CM Template. We note that CM Instance can also be defined independently, and then lately show conformance to a specific CM Template. In the fourth step ($\mathcal{I} \triangleright \mathcal{T}$), the accredited lab verifies the consistency between $\mathcal{I}$ and $\mathcal{T}$. This consistency check is crucial to build a chain of trust suitable for the cloud. Finally, in the fifth step (*CM Instance Execution*), the certification authority and the accredited lab execute all certification activities aimed to first certificate issuing (*issuing phase*) and then certificate adaptation (*post-issuing phase*). In the post-issuing phase, the CM Instance adapts itself to changes at both cloud (e.g., due to service migration), service (e.g., due to service versioning), and certification methodology (e.g., due to CM Template versioning) levels, ensuring (if possible) $\mathcal{I} \triangleright \mathcal{T}$ and verifying certificate validity. We note that the proposed certification process accomplishes cloud peculiarities. It supports *i)* the evaluation of a *multilayer ToC*, *ii)* a *dynamic* and *incremental* approach to certificate adaptation based on CM consistency check ($\mathcal{I} \triangleright \mathcal{T}$), and *iii) trust by delegation* providing a dynamic and runtime certificate life cycle management in the cloud.

3.4 Model Consistency Check

The consistency check $\mathcal{I} \triangleright \mathcal{T}$ is the main pillar of a certification process, since it verifies whether a CM Instance $\mathcal{I}$ is a correct and valid refinement of a CM Template $\mathcal{T}$, and represents the basis for building a chain of trust based on our certification process in Section 3.3. It is mandatory both when $\mathcal{I}$ is generated independently by cloud/service provider owning the ToC, and in case it is generated with the support of the certification authority or one of its accredited labs. The check that a CM instance $\mathcal{I}=\langle \overline{p}, \overline{ToC}, mi, \overline{ev}, \overline{l} \rangle$ is a valid instance of $\mathcal{T}=\langle p, ToC, m, ev, l \rangle$ is a matching process composed of 5 verification steps as follows.

3.4.1 Property and ToC verification

Non-functional property and ToC are strictly related elements, which are often used to index and collect certification requirements. Their verification proceeds as follows.

- Non-functional property verification (step 1): $\overline{p}$ is a valid instance of p, denoted $p \xrightarrow{I} \overline{p}$, *iff* $p \preceq_P \overline{p}$ on the basis of the hierarchy of properties $\mathcal{H}_P$ (see Section 3.2). Property verification checks that the property to be certified $\overline{p}$ is equal to/stronger than p.
- ToC verification (step 2): $\overline{ToC}=(\overline{\Theta},\overline{b})$ is a valid instance of $ToC=(\Theta,b)$, denoted $ToC \xrightarrow{I} \overline{ToC}$, *iff i)* $\forall \theta_j \in \Theta$, $\exists \theta_k \in \overline{\Theta} : \theta_j \preceq_{M_i} \theta_k$ and $\{events\}_{\theta_j} \subseteq \{events\}_{\theta_k}$, for $j=1,\ldots,|\Theta|$, $k=1,\ldots,|\overline{\Theta}|$ (see Section 3.2), *ii)* $\overline{b}=b$. ToC verification checks that *i)* for each mechanism specified in the ToC of CM Template, there exists the same or stronger mechanism in the CM Instance having a superset of annotated events and *ii)* the certificate binding layer is the same.

Example 3.4.1 *Let us consider a CM Template for payment systems with p=(*Confidentiality*, {*`ctx`=in-transit*}) and ToC=({{*encryption*,* `level`=message-in-transit*}},* <service>*), and a CM Instance for a specific payment system deployed on top of OpenStack with* $\overline{p}$*=(*Confidentiality*, {*`ctx`=in-transit*}) and* $\overline{ToC}$*=({*encryption*,* `algo`=XML-encryption*,* `protocol`=WS-Security*,* `level`=message-in-transit*},* <service>*).* $p \xrightarrow{I} \overline{p}$ *because* $p=\overline{p}$*;* $ToC \xrightarrow{I} \overline{ToC}$ *because the encryption mechanism based on XML-encryption in* $\overline{ToC}$ *is a refinement of the generic encryption mechanism for message in transit in ToC, and they both have a* service *binding.*

3.4.2 Evidence collection model verification

We aim to verify the consistency between the STS-based models $m \in \mathcal{T}$ and $mi \in \mathcal{I}$ (step 3). Our verification is successful, meaning that mi is a valid instance of m, if the linear independent paths modeling the testing flows $\Phi(mi)$ in the CM Instance $\mathcal{I}$ are equal to the flows $\Phi(m)$ in the CM Template $\mathcal{T}$. Our verification starts by defining the quotient graphs $G^m(V^m,E^m)$ and $G^{mi}(V^{mi},E^{mi})$ of models m and mi, respectively. Let us consider model $m=\langle \mathcal{S},s_0,\mathcal{V},\mathcal{I},\mathcal{A},\rightarrow \rangle$. The quotient graph is calculated on the basis of the equivalence relationship $\approx$, which models the belonging to the same functional mechanism $\theta \in \Theta$ in ToC, on the set of states $\mathcal{S}$ of m. The quotient graph $G^m(V^m,E^m)$ with respect to $\approx$ is a graph whose vertex set is the quotient set $V^m=S/\approx$ and two equivalence classes $[u],[v] \in V^m$ form an edge $([u],[v])$, iff $(u,v) \in \rightarrow$. We note that $\approx$ summarizes a set of consecutive or parallel functional mechanisms in mi as a single mechanism. This is due to the fact that mi specifies a variety of functional-related mechanisms that cannot be known a priori by m.

Given the quotient graphs G^m and G^{mi}, mi is a valid instance of m, denoted $m \xrightarrow{I} mi$, *iff* G^{mi} is isomorphic to G^m, as formalized by the following definition.

Definition 3.4.1 **(**$m \xrightarrow{I} mi$**)** *Let* $G^m(V^m,E^m)$ *be the quotient graph of* m*,* $G^{mi}(V^{mi},E^{mi})$ *be the quotient graph of* mi*. Also, let* θ_i *be the mechanism associated with each* $v_i \in V^m \cup V^{mi}$*.* mi *is a valid instance of* m*, denoted* $m \xrightarrow{I} mi$*, iff there exists an isomorphism* $f:V^m \rightarrow V^{\overline{m}}$*, such that the following conditions hold:*

1. $\forall n_i \in V^m$, $\exists f(n_i) \in V^{\overline{m}} \wedge \theta_{n_i} \preceq_{M_j} \theta_{f(n_i)}$ *on the basis of mechanism hierarchy* $\mathcal{H}_{M_j}$*;*
2. $\forall (n_i,n_j) \in E^m$, $(f(n_i),f(n_j)) \in E^{\overline{m}}$.

Condition 1 states that each vertex n_i in the evidence collection model of the CM Template should have a corresponding vertex $f(n_i)$ in the evidence collection model of the CM Instance, such that the mechanism associated with n_i is an abstraction of the one associated with $f(n_i)$ based on the relevant hierarchy. Condition 2 states that each edge in G^m should have a corresponding edge in G^{mi}.

We note that different possible isomorphisms can be found between $\mathcal{I}$ and $\mathcal{T}$. This is due to the fact that the abstract definition of $\mathcal{T}$ (in terms of abstract mechanisms) links to

several possible instantiations on the basis of mechanism hierarchy $\mathcal{H}_M$. As an example, m can specify a set of states referring to a generic encryption mechanism. The latter indirectly refers to both 3DES- and AES-based encryption mechanisms, opening the door to different isomorphisms.

Example 3.4.2 *Let us consider a CM Template for payment systems having a model m for secure data exchange as a sequence of two functional mechanisms interleaved by a security mechanism for channel encryption, and a CM Instance for a specific payment system having a model mi for secure data exchange as a sequence of a functional mechanism for the selection and configuration of the service operation to perform, a security mechanism based on XML-encryption for secure communications, and two functional mechanisms for operation execution and history management. The quotient graph of mi keeps the first two mechanisms as they are and merges the two functional mechanisms at the end of mi in a single mechanism.* $m \xrightarrow{I} mi$ *because* G^m *and* G^{mi} *have both* i) *a functional mechanism at the beginning and the end of the graph, and* ii) *a security mechanism in the middle such that the mechanism in* G^m *is an abstraction of the one in* G^{mi} *according to Example 3.4.1.*

3.4.3 Evidence verification

Evidence verification (step 4) uses the result of the evidence collection model verification to index the evidence ev to be matched against $\overline{ev}$. This step is needed because m refers to a set of abstract mechanisms and maps on a set $\{ev_1,\ldots ev_n\}$ of possible evidence. Evidence ev and $\overline{ev}$ are composed of a set of test cases tc=$\{(\theta,Pr,In,EO,Po)\}$ (see Section 3.2). Also, evidence $\overline{ev}$ can contain test cases that refer to more mechanisms than the ones referred by test cases in ev, and the cardinality of $\overline{ev}$ is equal to/greater than the cardinality of ev. As already discussed, this is due to the fact that mi specifies a variety of functional-related mechanisms that cannot be known a priori by m. For this reason, *i)* m can summarize a set of consecutive or parallel functional mechanisms in mi in a single mechanism, and *ii)* each quintuple (θ,Pr,In,EO,Po) in $tc \in ev$, with θ=(*Functional*,{}), has the form ((*Functional*,{}),Pr,*any*,*any*,Po).

After ev is selected, evidence verification continues executing a test case matching function (denoted $\times$) that verifies the correspondence between two test cases as follows.

Definition 3.4.2 (Function $\times$) *Let* tc_r=$\{(\theta_z,\ Pr_z,\ In_z,\ EO_z,Po_z)\}$ *and* tc_t=$\{(\theta_j\ Pr_j,In_j,\ EO_j,\ Po_j)\}$ *be two test cases.* tc_r *corresponds to* tc_t *(denoted* $tc_r \times tc_t$*)* iff $\forall(\theta_z,Pr_z,\ In_z,\ EO_z,\ Po_z) \in tc_r$ *one of the following conditions holds:*

1. *if* $\theta_z \neq$(Functional,{}), $\exists!$ $(\theta_j,Pr_j,In_j,EO_j,Po_j) \in tc_t$ *s.t.*
   - $(\theta_k,Pr_k,In_k,EO_k,Po_k) \in tc_t$, *with* $k<j$, *already satisfied either Condition 1 or Condition 2;*
   - $\theta_z \preceq_M \theta_j$;

- $\forall i \in In_z$, $\exists i \in In_j$ *s.t.* $[In_z.i]=[In_j.i]$;
- $\forall eo \in EO_z$, $\exists eo \in EO_j$ *s.t.* $[EO_z.eo]=[EO_j.eo]$.
- $Pr_z=Po_j$;
- $Po_z=Po_j$;

2. *else,* $\exists$ *zero or more consecutive* $(\theta_s,Pr_s,In_s,EO_s,Po_s) \in tc_t$ *s.t.*

   - $(\theta_k,Pr_k,In_k,EO_k,Po_k) \in tc_t$, *with* $k<s$, *already satisfied either Condition 1 or Condition 2;*
   - $\theta_s=($Functional$,\{\})$.

Definition 3.4.2 verifies the correspondence between tc_r and tc_t. The correspondence is evaluated by analyzing each quintuple $(\theta_z,Pr_z,In_z,EO_z,Po_z) \in tc_r$ and $(\theta_j,Pr_j,In_j,EO_j,Po_j) \in tc_t$ according to Conditions 1 and 2. Condition 1 is applied to each quintuple $(\theta_i,Pr_i,In_i,EO_i,Po_i)$ in tc_r referring to a non-functional mechanism (i.e., $\theta_z \neq (Functional,\{\})$). It is satisfied if a corresponding quintuple $(\theta_j,Pr_j,In_j,EO_j,Po_j)$ in tc_t is found, which defines a specialization of θ_z (second bullet), a set of inputs (outputs) in the same equivalence classes of inputs (outputs) of ev_r (third and fourth bullets), and the same set of pre-conditions Pr and post-conditions Po (fifth and sixth bullets). Condition 2 applies to each quintuple in tc_r referring to a functional mechanism (i.e., $\theta_z=(Functional,\{\})$). Each quintuple in tc_r corresponds to one or more consecutive and functional quintuples in tc_t. We note that both conditions define a constraint (first bullet) guaranteeing that all quintuples are verified keeping their ordering in tc_r and tc_t.

Example 3.4.3 *Starting from Example 3.4.2, let us consider a test case* tc_i *in* $\mathcal{T}$ *that selects an operation, receives an encrypted request and decrypts it, and executes the operation according to the received request, and a test case* tc_j *in* $\mathcal{I}$ *that selects operation* payment *of a payment service, receives an encrypted request with the credit card details and decrypts it, executes the operation according to the received request, and adds the results of its execution to the history log.* $tc_i \times tc_j$ *according to Definition 3.4.2.*

Following Definition 3.4.2, evidence $\overline{ev} \in \mathcal{I}$ is a valid instance of evidence $ev \in \mathcal{T}$ as follows.

Definition 3.4.3 ($ev \xrightarrow{I} \overline{ev}$) *Let* $ev=\{tc_1,\ldots,tc_k\}$ *be an evidence of* $\mathcal{T}$ *and* $\overline{ev}=\{\overline{tc}_1,\ldots,\overline{tc}_n\}$ *be the evidence of* $\mathcal{I}$, *with* $n \geq k$. $\overline{ev}$ *is a valid instance of ev, denoted* $ev \xrightarrow{I} \overline{ev}$, iff $\forall$ *path* $\overline{\phi}_i$ *of mi and corresponding path* ϕ_j *of m according to Definition 3.4.1,* $\forall tc_r \in ev$ *associated with* ϕ_j, $\exists tc_t \in \overline{ev}$ *associated with* $\overline{\phi}_i$ *s.t.* $tc_r \times tc_t$.

Definition 3.4.3 verifies the correspondence between each test case tc_r in CM Template and at least one test case tc_t in CM Instance. The correspondence is evaluated by analyzing all $(\theta_i,Pr,In_i,EO_i,Po) \in tc_r$ and $(\overline{\theta_j},\overline{Pr},\overline{In_j},\overline{EO_j},\overline{Po}) \in tc_t$ according to Definition 3.4.2.

3.4.4 Life Cycle verification

The certificate life cycle describes the expected evolution of a certificate over time, according to different events (e.g., unexpected testing failures, new version of a ToC). The lifecycle $\bar{l}$ in $\mathcal{I}$ must then adhere to the abstract lifecycle l in $\mathcal{T}$. $\bar{l}$ is a valid instance of l, denoted $l \xrightarrow{I} \bar{l}$, *iff* $G^{\bar{l}}(V^{\bar{l}},E^{\bar{l}})$ is isomorphic to $G^{l}(V^{l},E^{l})$ and each condition labeling edges in $\bar{l}$ is more restrictive of the corresponding in $\bar{l}$, as formalized by the following definition (step 5).

Definition 3.4.4 ($l \xrightarrow{I} \bar{l}$) *Let $G^{l}(V^{l},E^{l})$ be the lifecycle in $\mathcal{T}$ and $G^{\bar{l}}(V^{\bar{l}},E^{\bar{l}})$ be the lifecycle in $\mathcal{I}$. $G^{\bar{l}}(V^{\bar{l}},E^{\bar{l}})$ is a valid instance of $G^{l}(V^{l},E^{l})$, denoted $G^{l}(V^{l},E^{l}) \xrightarrow{I} G^{\bar{l}}(V^{\bar{l}},E^{\bar{l}})$,* iff *there exists an isomorphism $f{:}V^{l} \rightarrow V^{\bar{l}}$, such that:*

1. $\forall n_i \in V^{l}$, $\exists f(n_i) \in V^{\bar{l}} \wedge label(n_i)=label(f(n_i))$;
2. $\forall (n_i,n_j) \in E^{l}$, $(f(n_i),f(n_j)) \in E^{\bar{l}}$ *and is such that* $cond_{(f(n_i),f(n_j))}$ *is more restrictive than* $cond_{(n_i,n_j)}$.

Condition 1 states that each vertex n_i in G^{l}, representing a state of a certificate, should have a corresponding vertex $f(n_i)$ in $G^{\bar{l}}$ with the same label. Condition 2 states that each edge in G^{l} should have a corresponding edge in $G^{\bar{l}}$ with a more restrictive transition condition. In the following, for simplicity, we require each edge in G^{l} and corresponding edge in $G^{\bar{l}}$ to have the same condition, meaning that $\mathcal{T}$ and $\mathcal{I}$ must have the same lifecycle (e.g., the one in Figure 3.1). Existing approaches for the comparison of boolean expressions (e.g., [115]) can be integrated within lifecycle verification.

3.4.5 Full Model Consistency Check

Consistency check $\mathcal{I} \triangleright \mathcal{T}$ can then be modelled as a process composed of the 5 verification steps in this section. It is defined as a function $f^{\triangleright}{:}\mathcal{I}\times\mathcal{T}\rightarrow R$, where R models differences between $\mathcal{I}$ and $\mathcal{T}$ if $\mathcal{I} \not\triangleright \mathcal{T}$, $R{=}\emptyset$ otherwise, as follows.

Definition 3.4.5 ($f^{\triangleright}$) *Let $\mathcal{T}$ be a CM Template and $\mathcal{I}$ be a CM Instance, function $f^{\triangleright}{:}\mathcal{I}\times\mathcal{T}\rightarrow$*R *implements the CM Instance validity check $\triangleright$ and is such that:*

1. $f^{\triangleright}{=}\emptyset$ *($\mathcal{I} \triangleright \mathcal{T}$) iff* $p \xrightarrow{I} \overline{p}$ *(*Non-functional property verification*),* $ToC \xrightarrow{I} \overline{ToC}$ *(*ToC verification*),* $m \xrightarrow{I} mi$ *(*Evidence Collection Model verification*),* $ev \xrightarrow{I} \overline{ev}$ *(*Evidence verification*),* $l \xrightarrow{I} \bar{l}$ *(*Lifecycle verification*);*
2. $f^{\triangleright}$ *returns the differences* R *between $\mathcal{I}$ and $\mathcal{T}$ ($\mathcal{I} \not\triangleright \mathcal{T}$), otherwise.*

We note that $f^{\triangleright}$ can also be used to check validity of a CM Instance $\mathcal{I}$ (CM Template $\mathcal{T}$, resp.) w.r.t. an adapted CM instance $\mathcal{I}'$ (CM Template $\mathcal{T}'$, resp.). We also note that

R could additionally specify the differences between certification models in terms of test cases that *i)* become invalid in $\mathcal{I}'$ or *ii)* are specified in $\mathcal{I}'$ only. For conciseness, an example of consistency check based on Examples 3.3.1 and 3.3.2 can be found in Appendix B. We finally note that the computational complexity of $f^{\triangleright}$ can easily raise to a level that becomes unmanageable in a real-time scenario like the one in Chapter 4. In Section 3.7, we therefore define and experimentally evaluate two main heuristics that make the consistency function manageable at the price of a reasonable decrease in the quality of the consistency check.

3.5 Certificate Life Cycle Management

Certificate life cycle management relies on evidence collection and CM Instance validity check *i)* to verify the validity of a CM Instance w.r.t. the corresponding CM Template for certificate issuing (Section 3.5.1) and *ii)* to monitor changes to either CM Templates or CM Instances for certificate adaptation as discussed in the following.

3.5.1 Certificate Issuing

Certificate issuing first verifies the validity of a CM Instance w.r.t. the corresponding CM Template and is then managed according to transition $e_{NI,I}{=}NI{\rightarrow}I$ (dashed node/arrow in Figure 3.1). The transition is triggered, if corresponding condition $cond_{e_{NI,I}}$ of $e_{NI,I}$ in $\bar{l}$ is satisfied. The issuing phase is usually executed in a laboratory environment (pre-deployment) under the supervision of a certification authority, though in some specific situations it can be executed also in production by the delegated accredited lab (e.g., for properties that must evaluate systems in production such as availability via replica). When the certificate reaches the issuing state, life cycle management enters the second phase where the continuous execution of $\mathcal{I}$ triggers certificate adaptation.

3.5.2 Certificate Adaptation

Certificate adaptation is built around states I, S, R, and E of $\bar{l}$ in Figure 3.1. A certificate goes to state S when the collected evidence becomes insufficient to prove the property in the certificate; to state R when collected evidence is contradictory and does not prove the property (e.g., a successful attack is observed); to state E when the validity date expires. It returns from state S to state I when the collected evidence becomes sufficient for certificate re-new. We note that states R and E are final states and trigger re-certification from scratch. We also note that $cond_{e_{S,I}}{=}cond_{e_{NI,I}}$ with $e_{S,I}{=}S{\rightarrow}I$ and $e_{NI,I}{=}NI{\rightarrow}I$.

Certificate adaptation aims to maximize certificate validity, while minimizing the risk of unnecessary certificate revocation and reducing as much as possible the amount of re-certification activities. A certificate revocation in fact requires re-certification from scratch, which introduces high cost and time overheads invalidating the benefit introduced by cloud

Table 3.2 Adaptation summary

Scenario	**Adapted element**	$\triangleright$	$f^{\triangleright}$	**Adaptation actions**
CM Instance Adaptation ($\mathcal{I}\rightarrow\mathcal{I}'$)	$\overline{p}$	$\mathcal{I}'\triangleright\mathcal{T}$	$p\overset{I}{\rightarrow}\overline{p}'$	–
		$\mathcal{I}'\not\triangleright\mathcal{T}$	$p\overset{I}{\rightarrow}\overline{p}'$	Full re-certification
	$\overline{ToC}$	$\mathcal{I}'\triangleright\mathcal{T}$	$ToC\overset{I}{\rightarrow}\overline{ToC}'$, $m\overset{I}{\rightarrow}\overline{m}'$, $ev\overset{I}{\rightarrow}\overline{ev}'$	Partial re-evaluation
		$\mathcal{I}'\not\triangleright\mathcal{T}$	$ToC\overset{I}{\rightarrow}\overline{ToC}'$, $m\overset{I}{\rightarrow}\overline{m}'$, $ev\overset{I}{\rightarrow}\overline{ev}'$	Partial re-certification/ Upgrade/Downgrade
	$\overline{m}$	$\mathcal{I}'\triangleright\mathcal{T}$	–	–
		$\mathcal{I}'\not\triangleright\mathcal{T}$	$m\overset{I}{\rightarrow}\overline{m}'$, $ev\overset{I}{\rightarrow}\overline{ev}'$	Partial re-certification/ Upgrade/Downgrade
	$\overline{ev}$	$\mathcal{I}'\triangleright\mathcal{T}$	$ev\overset{I}{\rightarrow}\overline{ev}'$	Partial re-evaluation
		$\mathcal{I}'\not\triangleright\mathcal{T}$	$ev\overset{I}{\rightarrow}\overline{ev}'$	Partial re-certification/ Upgrade/Downgrade
	$\bar{l}$	$\mathcal{I}'\triangleright\mathcal{T}$	$l\overset{I}{\rightarrow}\bar{l}'$	–
		$\mathcal{I}'\not\triangleright\mathcal{T}$	$l\overset{I}{\rightarrow}\bar{l}'$	–
CM Template Adaptation ($\mathcal{T}\rightarrow\mathcal{T}'$)	Any	$\mathcal{I}\triangleright\mathcal{T}'$	All	CM Instance Adaptation (partial re-evaluation)
		$\mathcal{I}\not\triangleright\mathcal{T}'$	All	CM Instance Adaptation (partial re-certification)

certification schemes. We consider two adaptation scenarios: *i) CM Instance adaptation* reacting to a new version of service, platform, or infrastructure, or to any change in the configurations (e.g., due to elastic scaling, migration) at all cloud layers specified in the *ToC*; *ii) CM Template adaptation* reacting to new requirements for the validity of a property (e.g., a new bug in a mechanism or a new attack discovered). We note that any change in CM Template also triggers an adaptation process on CM Instance. Certificate adaptation is carried out through an incremental certification process. The incremental process provides the ability to re-execute (part of) the process in Figure 3.3, following changes in the CM Template, the CM Instance, and the service implementation. It re-validates the ToC according to the minimum set of test cases identified by validity check function $f^{\triangleright}$ in Section 3.4.

Table 3.2 shows a summary of the two adaptation scenarios. It describes the amount of verification activities (i.e., column $\triangleright$ and $f^{\triangleright}$) to be done on the basis of the adapted element (column *adapted element*), and the adaptation actions to be executed (column *adaptation actions*).

CM Instance Adaptation Let us consider an adapted CM Instance $\mathcal{I}'=\langle\overline{p}',\overline{ToC}',\overline{m}',\overline{ev}',\bar{l}'\rangle$ of $\mathcal{I}=\langle\overline{p},\overline{ToC},\overline{m},\overline{ev},\bar{l}\rangle$. CM Instance adaptation is triggered when $f^{\triangleright}(\mathcal{I}',\mathcal{I})\neq\emptyset$ and follows four different approaches.

Partial re-evaluation. It applies when $\mathcal{I}'\triangleright\mathcal{T}$, and considers transitions $e_{I,S}=I\rightarrow S$, $e_{S,I}=S\rightarrow I$, and $e_{S,R}=S\rightarrow R$. Partial re-evaluation first triggers transition $e_{I,S}$. It then executes an incre-

mental evidence collection process, which follows the differences between $\mathcal{I}$ and $\mathcal{I}'$ returned by $f^{\triangleright}(\mathcal{I},\mathcal{I}')$. It finally re-news the certificate ($e_{S,I}$), if possible, according to the following scenarios.

- *Cloud event.* A mechanism $\theta_i \in \overline{ToC}'$ of $\mathcal{I}$ is affected by the occurrence of an event. Test cases involving θ_i are re-executed according to the result of $f^{\triangleright}$.
- *Additional test cases.* They are added in $\mathcal{I}'$, due to a change in $\overline{ToC}'$, $\overline{m}'$, or $\overline{ev}'$, and executed.
- *New mechanism.* A new mechanism $\theta_j \in \overline{ToC}'$ of $\mathcal{I}'$ replaces a mechanism $\theta_i \in \overline{ToC}$ of $\mathcal{I}$ and is such that $\theta_j.\hat{\theta}=\theta_i.\hat{\theta}$ (i.e., they have the same type). All the test cases involving θ_i in the original CM Instance $\mathcal{I}$ are re-executed according to the new $\overline{ToC}'$ and $\overline{m}'$.

If enough correct evidence is collected the certificate returns to state I ($e_{S,I}$), otherwise partial re-certification can be triggered or the certificate can be revoked ($e_{S,R}$). We note that, in case a new life cycle $\overline{l}'$ is defined in $\mathcal{I}'$, no operations are needed *iff* $\overline{l}' \xrightarrow{I} \overline{l}$. We also note that *partial re-evaluation* does not require certificate authority intervention and can be executed at runtime following $\mathcal{I}'$.

Partial re-certification. It applies when $\mathcal{I}' \not\triangleright \mathcal{T}$, and considers transitions $e_{I,S}=I \rightarrow S$, $e_{S,I}=S \rightarrow I$, and $e_{S,R}=S \rightarrow R$. Partial re-certification first triggers transition $e_{I,S}$. It then executes an incremental evidence collection process that *i)* searches for a new $\mathcal{T}'$ such that $\mathcal{I}' \triangleright \mathcal{T}'$ and *ii)* follows the differences between $\mathcal{I}$ and $\mathcal{I}'$ returned by $f^{\triangleright}(\mathcal{I},\mathcal{I}')$. Partial re-certification exercises flows of $\mathcal{I}'$ that do not exist or are different from the ones in $\mathcal{I}$, rather than implementing a complete re-certification. It then executes new test cases to collect the evidence needed to award a certificate for a new property $\overline{p}'$ in $\mathcal{I}'$. If the evidence is sufficient and correct according to $\mathcal{T}'$, the certificate is re-newed ($e_{S,I}$). Otherwise the certificate is revoked ($e_{S,R}$). We note that, in case a new life cycle $\overline{l}'$ is defined in $\mathcal{I}'$, no operations are needed *iff* $\overline{l}' \xrightarrow{I} \overline{l}$.

Downgrade/Upgrade. It is a lightweight degeneration of the general case of partial re-certification that does not require new testing activities, at a price of a little involvement of the certification authority. Partial re-certification is executed only if downgrade/upgrade fail. Certificate *downgrade* is triggered when a set of test cases *i)* fail or *ii)* are removed from $\mathcal{I}'$ due to changes in $\overline{ToC}$ and/or $\overline{m}$. It aims to find a suitable CM template $\mathcal{T}'$ for the adapted CM Instance $\mathcal{I}'$, such that a weaker property is still preserved for the service referring to it. Templates for certificate downgrade are defined by the certification authority, making the accredited lab just responsible to check if $\mathcal{I}'$ is consistent with one of the alternative templates $\mathcal{T}'$. In case such $\mathcal{T}'$ is found, the original certificate $\mathcal{C}$ is downgraded to $\overline{\mathcal{C}}$. Certificate *upgrade* process is the inverse of the downgrade process and is only applicable to a downgraded certificate $\overline{\mathcal{C}}$, up to the original certificate $\mathcal{C}$. Downgrade and upgrade processes deal with

some classes of cloud configurations that change very rapidly (e.g., number of replicas supporting property availability).

Full re-certification. It is applied in case changes to $\mathcal{I}$ cannot be managed according to one of the above approaches.

3.5.3 CM Template Adaptation

CM Template adaptation focuses on incremental updates of the certification methodology. It is driven by the certification authority that releases a refined CM Template $\mathcal{T}'=\langle p',ToC',m',ev',l'\rangle$ of $\mathcal{T}=\langle p,ToC,m,ev,l\rangle$, and can trigger a CM Instance adaptation for all instances $\mathcal{I}$ referring to $\mathcal{T}$. The initial CM Template $\mathcal{T}$ is defined by the certification authority for a given property and class of ToC. However, upon new requirements for the validity of the property are discovered, the certification authority defines an adapted $\mathcal{T}'$ that is matched against $\mathcal{I}$ originally showing consistency with $\mathcal{T}$. The incremental process proceeds as follows: *i)* if $\mathcal{I}\triangleright\mathcal{T}'$, $f^{\triangleright}(\mathcal{T}',\mathcal{T})$ is performed and a partial re-evaluation executed according to its results; *ii)* if $\mathcal{I}\not\triangleright\mathcal{T}'$, the service under certification must be adapted and a new instance $\mathcal{I}'$ produced such that $\mathcal{I}'\triangleright\mathcal{T}'$. $f^{\triangleright}(\mathcal{I}',\mathcal{I})$ is then performed and a partial re-certification executed according to its results.

CM Template adaptation can be considered as a certification-aware fast-patching approach. As an example, suppose that United States Computer Emergency Readiness [4] identifies a new vulnerability for a given ToC, which calls for template $\mathcal{T}$ modification. Such modification triggers a top-down adaptation process, and all certificates referring to affected templates become *suspended*. A service owner must then adapt its own service and corresponding instance $\mathcal{I}$ to maintain the certificate.

3.5.4 Certificate Comparison

Traditional schemes, such as Common Criteria, have as one of the objectives allowing consumers to compare certified products on the market. The models presented in this thesis can be compared to understand which cloud service may best satisfy the customer security requirements. To provide such feature, certificates and certification model template and instance should be comparable. Certificate adaption scenarios showed that is possible to compare two different certification templates or instances starting from the assumption that $f^{\triangleright}$ is different than 0.

The easiest way to respectively compare templates, instances or certificates is by using the information provided by the certification property p. In fact p, as described in Section 3.2, is a pair (p,Ap) where p is an abstract property and Ap a set of attribute. In fact, given two properties $p_i,p_j\in\mathcal{P}$, p_i is weaker than p_j (denoted $p_i\preceq_P p_j$) if $p_i.\hat{p}=p_j.\hat{p}$, and $\forall a_k\in A_p$,

[4](US-Cert – https://www.us-cert.gov/)

either the value $n_i(a_k)$ of attribute a_k is not specified or $n_i(a_k) \preceq_{a_k} n_j(a_k)$. We note that total order relations $\preceq_{a_k}$ between contextual attributes $a_k \in A_p$ can be defined by expert users. An example of property hierarchy can be found in [4].

Certificates comparison is not a trivial task since they contain only part of the outcome of the whole certification process. However, the presented certification process allows user to evaluate in detail how the certificate was issued analyzing both its certification model instance and template supporting a deep investigation that traditional schemes, such Common Criteria, could not.

Certification model instances $\mathcal{I}$' and $\mathcal{I}$ can be easily compared if they are instances of the same template $\mathcal{T}$: $\mathcal{I} \triangleright \mathcal{T}$ and $\mathcal{I}' \triangleright \mathcal{T}$. In this case based on the $f^{\triangleright}$ function defined in 3.4.5 is possible to analyze the difference between the two CMI. The outcome of $f^{\triangleright}$ is R that describes the difference between $\mathcal{I}$ and $\mathcal{I}$'. The different is, if both CMI are instantiation of $\mathcal{T}$, based only on the soundness of the evidence and collection process requirements.

In case certification model instances $\mathcal{I}$' and $\mathcal{I}$ are not instantiation of a single $\mathcal{T}$, they could be compared *iff* $\mathcal{T}' \triangleright \mathcal{T}$ where $\mathcal{I}' \triangleright \mathcal{T}'$ and $\mathcal{I} \triangleright \mathcal{T}$. In this case we are sure that there is a consistency over the two models that guarantees a minimum set of overlapping characteristics. Even in this case the soundness of the models can be identified by analyzing the evidence collection model.

We note that due to the common structure of template and instance the function $f^{\triangleright}$ can be also defined as $f^{\triangleright}$: $\mathcal{T} \times \mathcal{T} \rightarrow R$ and $\mathcal{I} \times \mathcal{I} \rightarrow R$ and indeed the expression $\mathcal{T}' \triangleright \mathcal{T}$ is valid.

Example 3.5.1 *Let us consider two certification model instance $\mathcal{I}$ and $\mathcal{I}$' such that $\mathcal{I} \triangleright \mathcal{T}$ and $\mathcal{I} \triangleright \mathcal{T}$. We refer to the template $\mathcal{T}$ in example 3.3.1 where p=(*Confidentiality*, {*`ctx`*=*in-transit/at-rest*}) and target of certification ToC=({*θ_1, θ_2, θ_3*}, <*service*>). Both θ_1, θ_2, θ_3 are encryption mechanisms. The two CMI $\mathcal{I}$ and $\mathcal{I}$' are consistent with $\mathcal{T}$ by providing process to ensure encryption with a different requirement on the encryption algorithm used or on the accepted key strength. Based on this characteristic is possible to compare and evaluate the two CMI.*

3.6 Chain of Trust

The practical usability of a cloud certification process passes from the definition of a proper trust model enabling certification authorities to delegate part of the process management to accredited labs, and increasing the confidence of final users in the results of the certification process itself.

Cloud certification introduces the need to define a chain of trust where responsibilities are spread across the certification process life cycle and the entities involved in it. In fact, certification authorities cannot be assumed as a single trusted *CA* taking responsibility on (i.e., signing) the whole certification process. We therefore envision a chain of trust based on

multiple XML signatures. In the following, we denote with $A_{en,ws}$ assertions made by an entity en over a system/service ws, and with $E_{en,ws}$ the evidence produced by an entity en over ws and supporting $A_{en,ws}$. The customer c's trust in an assertion $A_{en,ws}$ made by an entity en is denoted $Tr(c, A_{en,ws})$, where Tr takes discrete values on an ordinal scale (e.g., for a Common Criteria [65] certified product, an assurance level value in 1-7).

The signing process at the basis of our chain of trust can be decomposed in three different signing moments, one for each of the components (CM Template, CM Instance, Certificate) of the certification process in Section 3.3, as follows.

- *CM Template signature*: CM Template $\mathcal{T}$ is signed by a trusted certification authority CA. It describes the methodology for the certification of a class of ToC, while it does not contain details about the evidence collection endpoints and the real ToC mechanisms. $Tr(AL,\mathcal{T}_{CA})$ denotes the trust an accredited lab AL has in CM Template $\mathcal{T}$ that builds on the trust AL has on CA and its signature.

- *CM Instance signature*: CM Instance $\mathcal{I}$ is signed by an accredited lab AL, which has been delegated by CA as the party responsible for instantiating the CM Template. AL receives a signed CM template $\mathcal{T}$ and fills in all missing elements (possibly with the help of the cloud/service providers) to form a CM Instance. The CM Instance signature builds on $Tr(AL,\mathcal{T}_{CA})$ and is at the basis of the trust $Tr(c, A_{AL,ws})$ and $Tr(c, E_{AL,ws})$ a client c has in assertions $A_{AL,ws}$ and evidence $E_{AL,ws}$, respectively, provided by AL.

- *Certificate signature*: this signature binds the certificate (including assertions $A_{AL,ws}$ and evidence $E_{AL,ws}$) and the corresponding CM Instance, which has been used to *i)* execute real testing activities on the target of certification and *ii)* produce the certificate itself.

Figure 3.4 shows our chain of trust, identifying roles (rectangles), artifacts (rounded rectangles), certification activities (solid arrows), and trust relations (dashed arrows). All signatures are implemented using public key cryptography. The chain of trust includes c's trust in *i)* CM Instance $\mathcal{I}_{AL,ws}$, denoted as $Tr(c,\mathcal{I}_{AL,ws})$, used to collect the evidence supporting a set of assertions, *ii)* the evidence generated by AL according to CM Instance $\mathcal{I}$, denoted as $Tr(c,E_{AL,ws})$, *iii)* assertions made by AL on a service, denoted as $Tr(c,A_{AL,ws})$, where $A_{AL,ws}$ is the set of assertions produced by the accredited lab AL on ws, and *iv)* the certificate $\mathcal{C}$ including $A_{AL,ws}$ and $E_{AL,ws}$. $Tr(c,\mathcal{C})$ depends on *i)* the reputation of CA signing CM Template $\mathcal{T}$ (i.e, $Tr(AL,\mathcal{T}_{CA})$) and the certificate $\mathcal{C}$ itself, *ii)* the reputation of AL and the trust in the methodology used by AL to generate and sign CM Instance $\mathcal{I}_{AL,ws}$ (i.e., $Tr(c,\mathcal{I}_{AL,ws})$), and specify assertions $A_{AL,ws}$(i.e., $Tr(c,A_{AL,ws})$), and *iii)* the trust in the methodology used by AL to generate evidence $E_{AL,ws}$ (i.e., $Tr(c,E_{AL,ws})$).

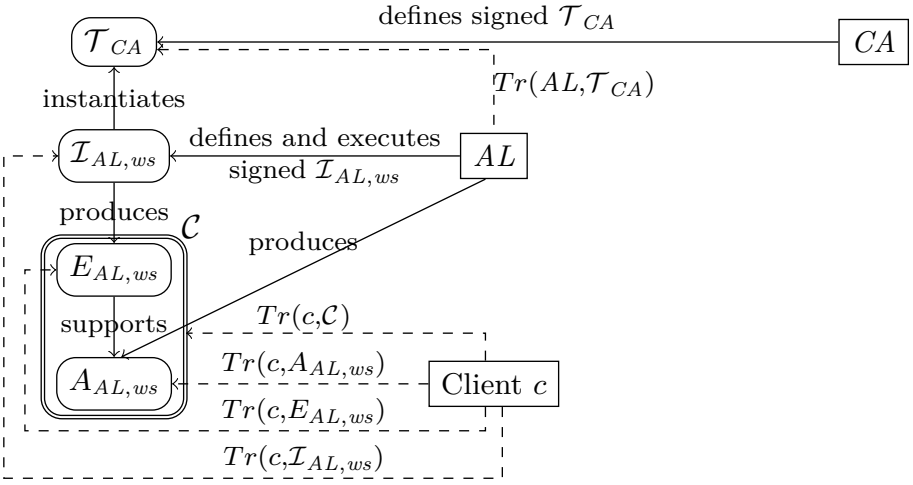


Fig. 3.4 Chain of Trust for cloud certification

3.6.1 Chain of Trust and Life Cycle Management

Our chain of trust supports the certificate life cycle in Figure 3.1, and both issuing and post-issuing phases.

Whenever issuing phase is concerned, there is a subtlety to consider. Since this phase is usually based on static evidence collected in a laboratory environment, the CM Instance signature must undergo a two-step process. The first process involves the signature of a CM Instance where the endpoints of the service under certification refer to mechanisms deployed in the laboratory environment. Upon a certificate $\mathcal{C}_{ws}$ is issued and certified service *ws* moved in production (i.e., there is a transition from state *NI* to state *I* in the life cycle), a second process substitutes the CM Instance, which is linked in the certificate, with a new one signed by *AL* with all bindings and endpoints referring to the real deployment infrastructure.

The chain of trust also considers post-issuing phase, where a certified system evolves to a new version or cloud events affecting it are observed. In this phase, as discussed in Section 3.5.2, the collected evidence may become insufficient or contradictory, and corresponding certificate invalid, requiring re-certification. The simplest approach is to always perform re-certification from scratch (i.e., certificate is revoked and the process starts from state *NI*); however, this approach introduces substantial time and cost overheads, which are not manageable in a highly dynamic cloud-based ecosystem. An incremental certification process producing evolving certificates, though more complex, is more adequate to the considered environment. Its main goal is to renew a certificate by reusing, as much as possible, the certification evidence available from older certificates [62], limiting collection of new evidence. Trust in an incremental process is given by the trust $Tr(c,E_{AL,ws})$ the client c has in the dynamic evidence produced by executing CM Instance $\mathcal{I}_{AL,ws}$ and the trust $Tr(c,\mathcal{I}_{AL,ws})$ c has in the instance itself. The evolving certificate generated as a result of the incremental process is

managed through our life cycle. For instance, as soon as the evidence is no more sufficient or part of it becomes invalid, the certificate is moved to state S, where AL evaluates if the certificate can evolve or not. In the first case, if the collected evidence is sufficient, the certificate comes back to state I. The involvement of the certification authority is marginal, since it only needs to sign the adapted certificate when required by accredited lab AL. The accredited lab in fact has been delegated by the certification authority, which trusts and verifies lab activities by means of digital signature verification. In the second case, the CM Instance is no more usable for service certification or compliance with the original CM Template cannot be guaranteed. Re-certification from scratch is then triggered.

3.7 Experimental Evaluation

We present a complete experimental evaluation of our approach for model consistency check, see Section 3.4, in terms of efficiency and quality, and discuss the utility and practical usability of our approach for life cycle management. The consistency check module is a pluggable module that can extend framework described in Chapter 5, by providing functionalities for template instance consistency check. Section 3.7.1 describes the heuristic we developed according to definition 3.4.5. All experiments have been repeated 3 times and the results shown in this section are the average over the 3 executions. All building blocks of our experiments, including the consistency check module, are available at Appendix B.

3.7.1 Consistency Check Algorithms

We implemented the *exhaustive algorithm* of our consistency check function $f^{\triangleright}$ as 5 consecutive verification steps according to Definition 3.4.5. We called it exhaustive because the evidence collection model verification is carried out by exhaustively searching if, among all possible permutations of flows $\Phi(mi)$ in $\mathcal{I}$, there exist one or more isomorphisms (Definition 3.4.1) with flows $\Phi(m)$ in $\mathcal{T}$. Considering β as the cardinality of $\Phi(m)$ in $\mathcal{T}$, the evidence collection model verification, and in turn the exhaustive algorithm, has a factorial asymptotic behavior $O(\beta!)$ in the worst case. The other 4 steps of $f^{\triangleright}$ show instead a polynomial behavior. We then propose two heuristics[5] balancing efficiency and quality in terms of precision and recall, which differs from the exhaustive algorithm only for the evidence collection model verification as follows. Heuristics pseudocode is available at Appendix B.

Heuristic 1: k-matching.

Evidence collection model verification is carried out flow by flow and aims to find multiple matching, isomorphisms in Definition 3.4.1, between m of $\mathcal{T}$ and mi of $\mathcal{I}$. It logically traverses

[5]Heuristics pseudocode is available in Appendix B

the permutation tree of the flows $\phi_j \in \Phi(mi)$ of $\mathcal{I}$ with a breadth-first search, and selects a proper sub-tree according to k and flows $\phi_i \in \Phi(m)$ of $\mathcal{T}$. k represents the maximum number of matching flows that are selected at each step of the heuristic. First, a node j at depth $d=1,\ldots,\beta$ in the permutation tree, with β the cardinality of $\Phi(m)$, is traversed *iff* its parent has less than k selected children in the sub-tree; then, it is selected *iff* $\phi_j \in \Phi(mi)$ matches the corresponding $\phi_d \in \Phi(m)$ according to Definition 3.4.1. The resulting sub-tree includes zero or more isomorphisms between m and mi, represented as paths of length β. In the worst case scenario, the algorithm has an exponential asymptotic behavior $\mathrm{O}(k^{\beta-k+1} \cdot (k-1)!)$, which for $k=\beta$ degenerates to the exhaustive algorithm $O(\beta!)$. We note that, for small k, the complexity is far lower than the one of exhaustive algorithm.

Heuristic 2: Ordered k-matching.

Evidence collection model verification is carried out by first ordering the flows in $\Phi(m)$ and $\Phi(mi)$, and then applying *k-matching* heuristic. We use an ordering function that recursively compares nodes at the same depth d, with $d=1,\ldots,\beta$, from the ancestors to the leafs. For each d, only flows that have not been ordered yet according to the previous runs of the ordering function are considered. The ordering function is based on the hierarchy of mechanisms $\mathcal{H}_M$ and given two flows ϕ_i and ϕ_j, with $i>j$, ϕ_i is placed first *iff* mechanism θ_i at depth d of ϕ_i and mechanism θ_j at depth d of ϕ_j are such that $\theta_j \prec \theta_i$. In the worst case scenario, the algorithm has the same asymptotic behavior as Heuristic 1, since the complexity of the ordering process is negligible compared to the one of *k-matching*.

3.7.2 Performance Evaluation

We evaluated the performance of our approach considering the matching between CM Templates and CM Instances at the basis of certificate lifecycle management. We automatically generated, using the tool available in Appendix B, 19 CM Templates $\langle p, ToC, m, ev, l \rangle$, varying the number β of flows between 1 and 19 (step 1), each with depth (i.e., flow length) equal to 5. Our tool selects a property p from the set of available properties, defines ToC ToC and life cycle l, and builds the evidence collection model m. Model m is composed of a set of β paths, implemented as flows of single mechanisms with depth 5. We note that a small depth equal to 5 has been chosen to demonstrate the high complexity of our matching approach also in simplified scenarios. For each template, we randomly generated 10 CM Instances (a total of 190 instances) that satisfy $f^{\triangleright}$, using the same tool available in Appendix B. Performance results measured consistency check verification between CM Templates and corresponding CM Instances in the worst case scenario, where all computations must be done to find a solution. Performance and quality experiments have been run on a VM with 22 cores, 16GB RAM, and 120GB HDD deployed on a physical machine Dell PowerEdge T620 equipped with 8 Xeon Octa Core 1.99 GHz.

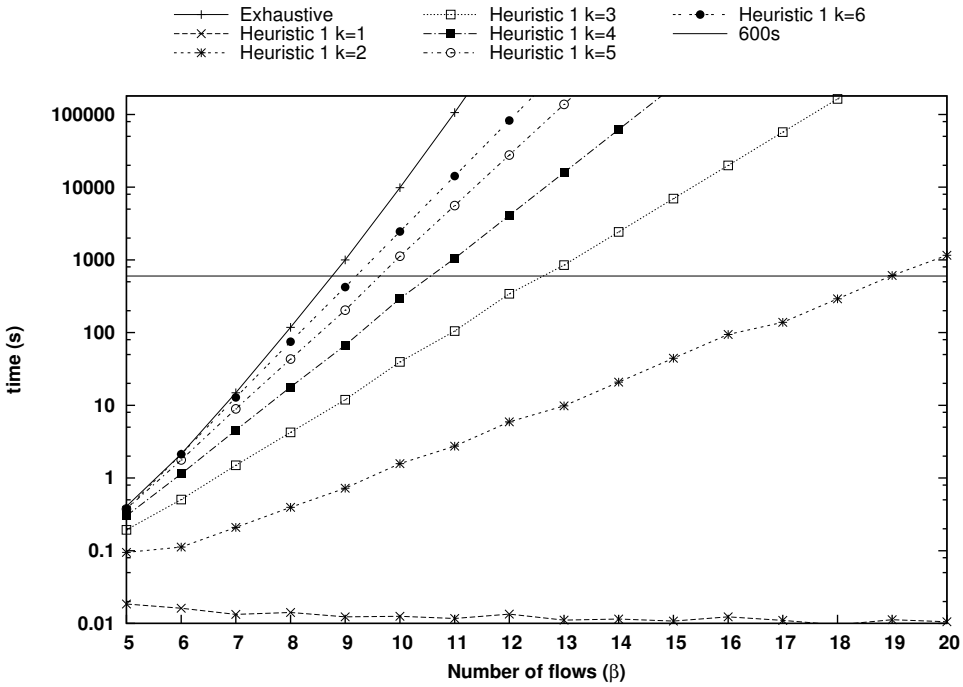


Fig. 3.5 Execution time (log scale) varying the number of flows

Figure 3.5 compares the execution time of the exhaustive algorithm and heuristic 1, using a log scale, varying k from 1 to 6, considering a fixed number of 50 test cases in evidence ev. Heuristic 2 has not been considered because it only adds a fixed delta for flow ordering. We note that the execution time of all algorithms is reported only for configurations requiring less than 10 minutes, and estimated for configurations over that threshold using the complexity analysis in Section 3.7.1. Our results show that, as expected, all heuristics approximates factorial execution time in the number of flows $\Phi(m)$, which can however be taken under control by selecting proper k. For instance, for k=3 execution time exceeds the 10-minute limit with $\Phi(m)$=13, for k=5 with $\Phi(m)$=10. The exhaustive algorithm shows the worst execution time, being k=β.

3.7.3 Quality evaluation

We evaluated the precision and recall of our heuristics with respect to the full precision and recall of our exhaustive algorithm, using three test sets. Each test set contains 160 consistent CM Instances derived from a single CM Template, with $\beta = 9$ and depth equal to 5. Each test set has an increasing number of average matching per flow, representing the mean number of flows in the 160 CM Instances matching a single flow in the CM Template. The first test set (a) is characterized by CM Instances having an average match per flow of 1.12 with variance 0.03; the second test set (b) has average match per flow of 2.26 with variance 0.25; the third test set (c) has average match per flow of 3.63 with variance 0.33. We note that the three

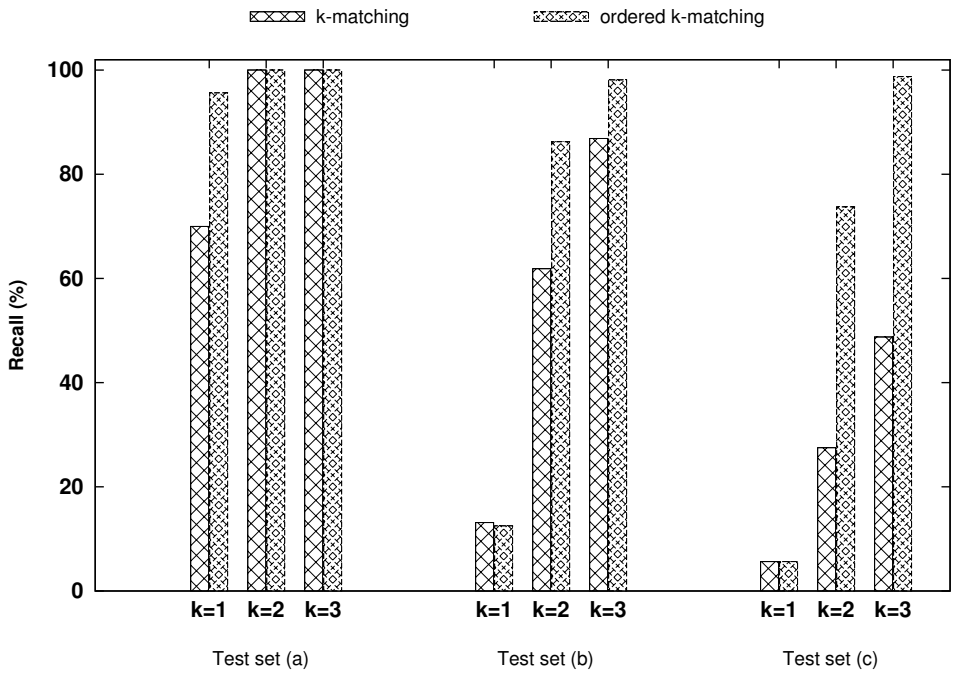


Fig. 3.6 Quality evaluation considering three instantiations of our k-matching algorithm with k=1, k=2, and k=3.

test sets model the three working of our consistency check function: *i)* the matching between two $\mathcal{I}$ (CM Instance adaptation - low average matching), *ii)* the matching between $\mathcal{T}$ and $\mathcal{I}$ (certificate issuing and CM Instance adaptation – medium average matching), and *iii)* the matching between two $\mathcal{T}$ (CM Template adaptation – high average matching).

First, we evaluated the recall of our heuristics on the three test sets, configuring our k-matching algorithm with three distinct values k=1, k=2, and k=3. Our results are presented in Figure 3.6. Heuristic 1 has 70% recall for test set (a), 13.125% recall for test set (b), and 5.625% recall for test set (c) with k=1; 100% recall for test set (a), 61.875% recall for test set (b), and 27.5% recall for test set (c) with k=2; 100% recall for test set (a), 86.875% recall for test set (b), and 48.75% recall for test set (c), with k=3. Heuristic 2 provides a substantial improvement for k=2 and k=3, which have 86.25% and 98.125% recall for test set (b), 73.75% and 98.75% recall for test set (c) respectively. Our results show that the ordering introduced in Heuristic 2 has a positive effect on the recall. This is due to the fact that the ordering, especially with k>1, increases the probability of correctly matching the evidence model in the template with the one in the instance. In addition, our results show that the higher k, the higher the recall. This is due to the fact that, with higher k, multiple evidence model matching between a template and an instance are found, increasing the probability of finding the one that also makes the whole consistency check successful.

Second, we evaluated the precision of our heuristics by introducing ad hoc random variations on all 480 CM Instances (e.g., modifications of flows, mechanisms, properties) in the three test sets, to produce CM Instances which are inconsistent with the corresponding CM Templates. We then executed our heuristics obtaining no matching, meaning that our heuristic algorithms do not produce any false positives or, in other words, an inconsistent CM Instance never shows consistency with a CM Template.

To conclude, although all heuristics approximate factorial execution time, high-quality results can be achieved with small k and good performance. For instance, ordered k-matching, with k=2, achieves quality of 86.25% on test set (b) with a worst case performance of 0.52s; with k=3, it achieves quality of 98.75% on test set (c) with a worst case performance of 0.7s.

3.8 Chapter Summary

The definition of a certification scheme for the cloud is an important research direction and a practical need. We have provided a rigorous and adaptive assurance technique based on certification which fully addresses cloud requirements. Above all, our certification scheme provides a solution to certificate life cycle management including an automatic and incremental approach to certificate adaptation addressing the multi-layer and dynamics nature of the cloud. Our scheme departs from the assumption of having an online certification authority always available during the certification process, and is at the basis of a concrete trust model for the cloud.

Chapter 4

Cloud Service Deployment based on non-functional properties

The maturity reached by cloud computing has fostered the implementation of a number of infrastructure, platform, and application services available worldwide. At the same time, the trend toward coarse-granularity business services, which cannot be managed by a single entity, resulted in several approaches to service composition that maximize software re-use [116]. Service compositions need to guarantee optimal and verifiable QoS, usually achieved by means of assurance techniques for the verification of non-functional properties (e.g., security, performance) [117, 118]. Recently, certification-based assurance techniques have been introduced to guarantee stable QoS in the cloud [119–123]. They are based on continuous collection of evidence on the behavior of a cloud-based system, which is used to verify whether the considered system holds a specific (set of) property. Certification techniques mostly focus on the certification of single-service, such as described in Chapter 3, and often do not consider the cost of maintaining stable QoS. Even worse, a trend in service composition is to provide an ad hoc composite service for each request, with high costs on the cloud providers (CPs).

Starting from the concepts expressed in Chapter 3, this Chapter[1] first analyzes and defines a certification approach for cloud composite service. Certification of composite service is a challenging task since certification is mostly seen as a monolithic process and indeed traditional approaches require to re-execute all certification activities for all services involved. In our view certification of composite services should minimize all these activities and re-use

[1]This chapter is based on the following publications:

- A Cost-Effective Certification-Based Service Composition for the Cloud, Co-author: M. Anisetti, C.A. Ardagna, E. Damiani, published in Proc. of IEEE CLOUD 2016 , June-July 2016, San Francisco, CA, USA
- Cost-Effective Deployment of Certified Cloud Composite Services, Co-author: M. Anisetti, C.A. Ardagna, E. Damiani, under review: Journal of Parallel and Distributed Computing

all available artefact such as certificates and corresponding evidence. A composite certificate, built on top of single-service-certifcates, is subject to changes of the involved services and indeed its life-cycle management must support versioning and migration. Moreover, due to the dynamic nature of the cloud, versioning and migration must be accomplished in a near real time to guarantee the certification continuity.

This chapter also faces the need to control the costs observed by cloud providers for certified composition management. In this scenario, it is important to provide a cost-effective certification for cloud composite services. While current research on cloud computing has privileged solutions minimizing costs on the final users neglecting the costs on the cloud providers that often represent the first source of fee increase. In this chapter we provide a deployment algorithm for cloud service composition based on non-functional properties that aims to minimize cloud provider's costs. Moreover, as highlighted by Lins et al. [13], the certification effectiveness is strongly dependant by cost. The approach presented in this chapter minimizes the certification costs, but without the main goal of favouring the cloud provider but with the aim to enhance the fruition of security cloud services. In fact the possibility to re-use certificates and automate the certification trust chain can minimize the involvement of actors such as CA and at the same time the number of effectively necessary audit operations.

4.1 Reference Model and Requirement

Our reference model is a cloud infrastructure where single services are composed to form complex services and certification-based assurance techniques are deployed for continuous QoS evaluation. The participating entities are: *i) cloud provider*, providing functionalities for service delivery and composition; *ii) composite service owner*, managing a service composition; *iii) certification authority*, providing functionalities for continuous QoS certification. Current approaches to service composition in the cloud are affected by few limitations, which show a clear disalignment with the maturity reached by the cloud. These limitations, which are described in the following, must be addressed to provide a cost-effective service composition for the cloud with continuous QoS assessment.

- *Functional composition.* Service composition in the cloud puts great emphasis on functionalities. Component services are selected on the basis of the implemented functionalities, while overall non-functional aspects are, in most of the cases, pushed aside. This practice however increases the likelihood of composite services that, on one side, satisfy the expectations of the users, while on the other side increase risks of failures and mishbehaviors. A proper approach to service composition must not only focus on functional requirements, but also consider non-functional requirements from the outset.

- *Ad hoc composite services.* Service composition in the cloud often consists of ad hoc workflows, where component services are designed and developed for a specific composite service. This approach substantially decreases the utility of service composition, both from a flexibility and a cost point of view. Having no possibility of sharing a single service among multiple service compositions bound current approaches to old fashion monolithic service deployments. This approach is often adopted at infrastructure layer, where the huge amount of available resources often point to single tenant scenarios, where a user asking for resources is usually provided with isolated resources not shared with other tenants. If, on one side, ad hoc composite services lower complexity of QoS evaluation and management, on the other side, it substantially increases costs and reduces the benefits of service composition.

- *QoS evaluation.* It mostly focuses on deployment-time cost evaluation and on composition adaptation in case of component service malfunctioning/failure. QoS evaluation is however a more powerful concept that should represents a first-class requirement driving composition operations. First, it should be based on assurance (e.g., certification) techniques guaranteeing stable and verifiable QoS; then, it should consider how the QoS of a single service contributes to the QoS of the whole composition; finally, it should implement a continuous process that evaluates QoS over time and drives adaptation of service compositions to provide stable QoS.

- *Costs.* The evaluation of service composition costs, which mainly focuses on direct costs due to component service integration, does not fit a multi-tenant cloud environment where *i)* services can be shared, relocated and migrated among different compositions, and *ii)* non-functional properties are modeled as QoS requirements and integrated with the composite service life cycle. A proper cost evaluation and cost-effective service composition must also consider costs introduced by the infrastructure responsible for continuous QoS evaluation, and the costs introduced by the mismatch between QoS requested by the users and the ones provided by the cloud infrastructure.

In the following of this chapter, we provide a cost-effective, certification-based service composition approach for the cloud that fills in the above limitations. It is based on *i)* the concept of *portable certification*, supporting certification of composite cloud services continuous providing QoS evaluation also in case of service migration and relocation (Section 4.2), and *ii)* a new cost evaluation methodology, considering direct, indirect, and mismatch costs on the cloud providers (Section 4.3, 4.4).

We note that this chapter uses light versions of the concepts described in Chapter 3, since the requirements of time and cost effectiveness of the process can't be match using the whole definition of $\mathcal{I}$ and $\mathcal{T}$ and mainly of consistency check $f^{\triangleright}$ described in Section3.4. $f^{\triangleright}$ is NP-Hard and in this Chapter we mainly focus not on the consistency of two models but

on how effectively deployed cloud services minimizing their cost, indeed we cutted some of the characteristics of the models to make the chapter as much readable as possible, without loss of generality. Section 4.2 describes in details the differences.

4.2 Certification of Cloud Composite Services

We present a certification approach specifically tailored for cloud composite services, which is grounded on and extends the one in Section 3.3 to *i)* support service versioning, migration, and deployment changes (portability) and *ii)* accomplish the dynamics of service orchestrations where component services can be replaced and migrated at run time according to contextual events.

In this chapter, without loss of generality, we consider a lighter version of the concepts described in Chapter 3. We still comply with the whole certification process, but due to the requirements of the deployment problem, which should support versioning, migration and portability in near real time manner to guarantee the certification continuity, we cut some of the characteristics that make the consistency check (see Section 3.4) a NP-hard problem. In fact, our certification process is still driven by a Certification Authority that manages all certification activities leading to certification. It is composed of two sub-processes: *i) evidence collection sub-process* and *ii) life cycle sub-process*. The *evidence collection sub-process* collects the evidence at the basis of a trustworthy certification and is carried out by the certification infrastructure. The *life cycle sub-process* implements a continuous certification process that accomplishes the evolution of the ToC, managing ToC migrations and versioning. The cloud service under evaluation is referred to as *Target of Certification* (ToC). Properties $p{=}(\hat{p},l)$, as defined by the Cloud Security Alliance [112], are composed of a controlled name $\hat{p}$ (e.g., confidentiality of data in transit) and a level l modeling the strength of the supported property. Properties can be organized in a hierarchy based on their strength such that $p_i{\leq}p_j$ (meaning p_i is weaker than p_j) *iff* $p_i.\hat{p}{=}p_j.\hat{p}$ and $p_i.l{<}p_j.l$. Based on levels l, a distance $Dist(p_i, p_j)$ between two properties with the same $\hat{p}$ is defined as:

$$Dist(p_i, p_j) = |p_i.l - p_j.l| \tag{4.1}$$

4.2.1 Certification Model Template and Instance

Before discussing the two subprocesses, we define our certification model template and instance.

Certification Model Template ($\mathcal{T}$). It is a declarative model that describes the activities to be done to verify a set of properties according to the expected behavior of a class of ToC. Formally, a CM Template $\mathcal{T}_i$ is a triple (f_i, R_i,*d-eval*$_i$), where *i)* f_i is a functionality in the set F of functionalities offered by a cloud provider, *ii)* r_k is a user requirement in the set R_i of

requirements used to annotate f_i, with $r_k \in R_i$ a property $(\hat{p},l)$, and *iii) d-eval*$_i$ is a declarative description of the evaluation activities to be carried out on the *ToC* to verify requirements R_i. $\mathcal{T}$ is built around *d-eval*, which is defined as a set of annotated workflows.

Definition 4.2.1 (*d-eval*) d-eval *is a pair* $\langle \phi, \omega \rangle$*, where:*

- ϕ *is a set of sequential workflows* $\{n_1,\ldots,n_n\}$ *for evidence collection, where each node* n_i *defines an abstract action (e.g., test authentication interface) and each edge* (n_i,n_j) *the flow between two actions.*
- ω *is an annotation function on nodes n.* $\omega(\{n_i\})$ *defines constraints (e.g., two factor authentication required) for a subset* $\{n_i\}$ *of abstract actions.*

We recall that *d-eval* refers to a generic class of *ToC* (e.g., an authentication system), while it precisely pinpoints security and deployment requirements (e.g., a given password strength policy). This means that, although there are a number of different *ToC* for the selected class, their evaluation w.r.t. security/deployment requirements should follow the same declarative description.

Certification Model Instance ($\mathcal{I}$). It is a procedural, executable model generated by instantiating $\mathcal{T}$ on a real *ToC*. It drives the certification process, including the evidence collection process. Formally, a CM Instance $\mathcal{I}_i$ is a triple $(cs_i, \mathcal{P}_i, p\text{-}eval_i)$, where *i)* cs_i is the *ToC*, *ii)* $\mathcal{P}_i$ is the set of properties supported by $\mathcal{I}_i$, and *iii)* $p\text{-}eval_i$ defines certification activities as a concrete instantiation of *d-eval* for a specific *ToC*. $\mathcal{I}$ is built around *p-eval*, which covers the peculiarities of the specific *ToC* w.r.t. the given properties. *p-eval* is an annotated workflow defined as follows.

Definition 4.2.2 (*p-eval*) p-eval *is a triple* $\langle \phi', \lambda \rangle$*, where:*

- ϕ' *is a set of sequential workflows* $\{n_1,\ldots,n_n\}$ *for evidence collection, where each node* n_i *defines an action implemented on the ToC instance and each edge* (n_i,n_j) *the flow between two implemented actions.*
- λ *is an annotation function.* $\lambda(\{n_i\})$ *defines the configuration settings of each action, describes how to deploy* p-eval*, and describes possible dependencies on its execution.*

We note that CM Instance $\mathcal{I}$ can be not unique for CM Template $\mathcal{T}$.

Example 4.2.1 *Let us consider a Certification Model Template* $\mathcal{T}$*=(*Storage*, Confidentiality via encryption at rest,* d-eval*), with* d-eval=$\langle \phi, \omega \rangle$*. For simplicity, we assume* ϕ *composed of a single sequential workflow* $\{n_1, n_2, n_3\}$*, where* n_1*="ToC login",* n_2*="Test encryption",* n_3*="ToC logout", and annotations* $\omega(\{n_1\})$*=[Administration credentials required],* $\omega(\{n_2\})$*=[Resource URI].*

The same Certification Model Template $\mathcal{T}$ is instantiated in two different Certification Model Instances $\mathcal{I}$ for a linux file system and Amazon S3. Both instances drive a certification process and evidence collection activity targeting the same property "Confidentiality at rest via encryption".

Let us first consider a linux file system using LUKS. p-eval$_l$=$\langle \phi_l^{'}, \lambda_l \rangle$ *implementing the above* d-eval *is defined as follows:* $\phi_l^{'}$=*{SSH login, Script testing encrypted volumes, SSH logout},* $\lambda_l(\{n_1\})$=*[root,cert],* $\lambda(\{n_2\})$=*Volume path. Let us then consider Amazon S3 Storage.* p-eval$_{s3}$=$\langle \phi_{s3}^{'}, \lambda_{s3} \rangle$ *implementing the above* d-eval *is defined as follows:* $\phi_{s3}^{'}$=*{Amazon login, API call for S3 configuration, Amazon logout},* $\lambda_l(\{n_1\})$=$[credentials, APIkey]$, $\lambda(\{n_2\})$=*[Config item].*

4.2.2 Certification Portability

In the following we first describe the portability of our certification process and then describe how we use it in the framework of certification of composite services.

A portable certification process is a certification process that is not bound to a specific *ToC* and can be easily applied to different service instances. In other words, it permits to apply the same certification process to different *ToC* with sufficient commonalities. Using our formalism, a certification process that derives from requirements in a specific template $\mathcal{T}$ can be re-used (with or without minor modifications) to certify all the services having an instance $\mathcal{I}$ consistent with $\mathcal{T}$. To verify this consistency we define a consistency check function, inspired by $f^{\triangleright}$ described in Section 3.4 , as follows.

Definition 4.2.3 ($\xrightarrow{I}$) *CM Instance* $\mathcal{I}_i$=($cs_i, \mathcal{P}_i$,p-eval$_i$) *is consistent with CM Template* $\mathcal{T}_i$=(f_i, R_i,d-eval$_i$)*, denoted as* $\mathcal{T}_i \xrightarrow{I} \mathcal{I}_i$*, iff* i) cs_i *implements* f_i, ii) $\mathcal{P}_i$ *is such that* $R_i \leq \mathcal{P}_i$*, that is,* $\forall r_j \in R_i, p_j \in \mathcal{P}_i$, $r_j \leq p_j$*, meaning that the properties are stronger than the requirements according to property levels, and* iii) d-eval$_i \xrightarrow{i}$p-eval$_i$ *(see Definition 4.2.4), meaning that* p-eval$_i$ *is an instantiation of* d-eval$_i$*.*

Consistency check $\xrightarrow{I}$ is the cornerstone of the process portability. A certification process can be implemented and executed using different instances $\mathcal{I}$, thanks to the decoupling between abstract definition ($\mathcal{T}$) and concrete actuation ($\mathcal{I}$) of the certification process. This decoupling also permits multiple consistent instantiations ($\mathcal{I}$) of the same process ($\mathcal{T}$). We note that, having $\mathcal{T}$ and $\mathcal{I}$ the same logical structure, $\xrightarrow{I}$ can be used to verify the consistency between two templates ($\mathcal{T}_i \xrightarrow{I} \mathcal{T}_j$) and two instances ($\mathcal{I}_i \xrightarrow{I} \mathcal{I}_j$).

As a complement to Definition 4.2.3, we detail how *p-eval* in $\mathcal{I}$ is checked for consistency against *d-eval* in $\mathcal{T}$.

Definition 4.2.4 ($\xrightarrow{i}$) p-eval$_i$=$\langle \phi', \lambda \rangle$ *is an instantiation of* d-eval$_i$=$\langle \phi, \omega \rangle$*, denoted as* d-eval$_i \xrightarrow{i}$p-eval$_i$*, iff* i) ϕ' *implements* ϕ*,* ii) *configurations* $\lambda(\{n_i\})$ *in* p-eval *instantiate*

constraints $\omega(\{n_i\})$ in d-eval*,* iii) *λ permits the binding between each action in ϕ' and the corresponding end-point in the ToC.*

Definition 4.2.3 ($\xrightarrow{I}$) and Definition 4.2.4 ($\xrightarrow{i}$) are at the basis of a portable certification process that addresses two main scenarios: *service versioning* and *service replacement.*

Service versioning. It considers a single service that is either migrated as is to another location or evolves to a new version. Process portability for service versioning is defined as described in the following definition.

Definition 4.2.5 (Process Portability (Versioning)) *Let us consider a certification process driven by $\mathcal{I}_i$=($cs_i, \mathcal{P}_i$,*p-eval$_i$*) for service cs_i, and a service cs_k such that either* i) *cs_i=cs_k but they are deployed in different locations or* ii) *cs_k is the new version of cs_i. The certification process driven by $\mathcal{I}_i$ can be ported to cs_k iff λ_i is modified to connect* p-eval$_i$ *to cs_k.*

Process portability (versioning) properly configures the certification model instance in a way that permits the certification activities in *p-eval$_i$* to connect to a different *ToC* (i.e., service cs_k). To this aim, λ_i of *p-eval$_i$* must provide the new configurations required to connect each action to cs_k.

Service replacement. It considers a migration of a service to another service of the same class. For instance, a service implementing a MySQL database is migrated to a service implementing an SQLServer database. Process portability for service replacement is defined as described in the following definition.

Definition 4.2.6 (Process Portability (Replacement)) *Let us consider $\mathcal{I}_i$=($cs_i, \mathcal{P}_i$,*p-eval$_i$*) and $\mathcal{I}_k$=($cs_k, \mathcal{P}_k$,*p-eval$_k$*) such that $cs_i \neq cs_k$. The certification process driven by $\mathcal{I}_i$ can be ported to $\mathcal{I}_k$ according to the following conditions:*

- *$\mathcal{I}_i \xrightarrow{I} \mathcal{I}_k$*
- *cs_i and cs_k provide the same functionality f_i.*

Process portability (replacement) instantiates certification activities on different services cs_i and cs_k. To this aim, Condition 1 states that $\mathcal{I}_i$ is consistent with $\mathcal{I}_k$, and in turn their $\mathcal{T}$ are consistent as well. We note that the consistency at CM Instance level implies that *p-eval$_k$* implements *p-eval$_i$* (see Definition 4.2.4). In other words the workflows for evidence collection in *p-eval$_i$* must be available also in *p-eval$_k$* possibly with different annotation functions [6].

Condition 2 states that cs_i and cs_k provide the same functionality f_i, which is specified in the corresponding templates $\mathcal{T}_i$ and $\mathcal{T}_k$. In other words, a certification process can be ported to a service or in an environment where the certification is driven by a different $\mathcal{T}$ without the need to re-build the certification process from scratch.

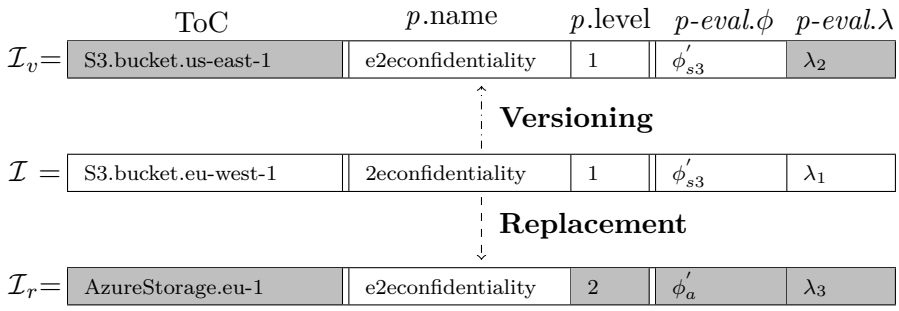


Fig. 4.1 An example of Versioning and Replacement of a storage service

Example 4.2.2 *Let us consider a CM template for a storage service defined as follows* $\mathcal{T}_1$*={Storage, {(e2econfidentiality,1) },* d-eval$_{storage}$*} where* end-to-end confidentiality *(e2econfidentiality) is requested (i.e., both confidentiality in transit and confidentiality at rest). Let us consider a storage service based on Amazon Simple Storage Service (S3) and specifically a bucket hosted on S3 eu-west-1 AWS region. Let us consider that this service has been certified according to the CM Instance* $\mathcal{I}_{s3}$*=(S3.bucket.ue-west,{(e2econfidentiality,1)},*p-eval$_{s3}$*) with* $\mathcal{T}_1 \xrightarrow{I} \mathcal{I}_{s3}$.

Figure 4.1(a) shows the case where this service is moved to a different region (versioning). In this scenario CM Instance $\mathcal{I}_{s3}$ *is ported to (S3.bucket.us-east-1,{(e2econfidentiality,1)},*p-eval$_{s3}$*), where p-eval*$_{S3}$ *is re-configured to access the new bucket in the different region. We note that only parameters* λ_2 *are modified since the service is exactly the same but in different location.*

Figure 4.1(b) shows the case where this service is migrated to a different service (replacement). More specifically the service is replaced with an Azure Storage service which offers the same functionality and is certified for a given $\mathcal{T}_2$ *for the same property but with an higher level ({(e2econfidentiality,2)}) with (*$\mathcal{T}_1 \xrightarrow{I} \mathcal{T}_2$*). The corresponding CM Instance* $\mathcal{I}_{As}$*=(AzureStorage.ue-1,{(e2econfidentiality,2)},*p-eval$_a$*) is compatible with* $\mathcal{I}_{s3}$ *and can replace it. We note that, in a real environment, storage service replacement also implies functional compatibility at orchestration level and application data migrations.*

4.2.3 Deployment Composition Matrix

A certification process for composite services builds on our portable certification process and is driven by a compositional CM Template $\hat{\mathcal{T}}$, where functional and certification requirements are specified for each component service. $\hat{\mathcal{T}}$ is expressed as a set $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ of ordered templates, each to be linked to a component service. A certified service cs_i, having $cert_{\mathcal{I}_i}$, can be selected as a component service *iff* its $\mathcal{I}_i$ is consistent with the corresponding $\mathcal{T}_i$ in $\hat{\mathcal{T}}$. We note that templates, including compositional templates, are specified by a CA and

are the cornerstone of the certification chain of trust described in Section 3.6. We extend Definition 4.2.3 to compositional instances ($\hat{\mathcal{I}}$) and compositional templates ($\hat{\mathcal{T}}$) as follows.

Definition 4.2.7 ($\xrightarrow{\hat{\mathcal{I}}}$) *A Compositional Instance $\hat{\mathcal{I}}i$ is consistent with a Compositional Template $\hat{\mathcal{T}}i$, denoted as $\hat{\mathcal{T}}i \xrightarrow{\hat{\mathcal{I}}} \hat{\mathcal{I}}i$, iff $\forall \mathcal{T}_k \in \hat{\mathcal{T}}i$, $\exists \mathcal{I}_j \in \hat{\mathcal{I}}i$ such that $\mathcal{T}_k \xrightarrow{I} \mathcal{I}_j$.*

The consistency check in Definition 4.2.7 supporting multiple consistent instantiations ($\hat{\mathcal{I}}$) of the same certification process ($\hat{\mathcal{T}}$) is at the basis of composition portability. Composition portability supports automatic component substitution and reuse, where each component service certified according to an instance $\mathcal{I}_i$, such that $\mathcal{T}_i \xrightarrow{I} \mathcal{I}_i$, can be orchestrated in the composition identified by $\hat{\mathcal{T}}$. The consistency check supports higher flexibility and lower costs. Shared/reused components do not need to be evaluated multiple times, saving certification effort, and their management does not involve the certification authority.

The status of a given CP at time t can be represented as a matrix D of size $C \times F$ of deployed compositions $\hat{\mathcal{I}}i$, where C is the cardinality of deployed compositions at time t and F the cardinality of all possible functionalities provided by service providers. Matrix D has the following structure

$$D = \begin{array}{c} \\ \hat{\mathcal{I}}1 \\ \hat{\mathcal{I}}2 \\ \hat{\mathcal{I}}3 \\ \vdots \\ \hat{\mathcal{I}}C \end{array} \begin{array}{c} \begin{array}{cccccc} f_1 & f_2 & f_3 & f_4 & \cdots & f_F \end{array} \\ \begin{pmatrix} \mathcal{I}_{1,1} & \mathcal{I}_{1,2} & \mathcal{I}_{1,3} & \mathcal{I}_{1,4} & \cdots & \mathcal{I}_{1,F} \\ \mathcal{I}_{2,1} & \mathcal{I}_{2,2} & \mathcal{I}_{2,3} & \mathcal{I}_{2,4} & \cdots & \mathcal{I}_{2,F} \\ \mathcal{I}_{3,1} & \mathcal{I}_{3,2} & \mathcal{I}_{3,3} & \mathcal{I}_{3,4} & \cdots & \mathcal{I}_{3,F} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathcal{I}_{C,1} & \mathcal{I}_{C,2} & \mathcal{I}_{C,3} & \mathcal{I}_{C,4} & \cdots & \mathcal{I}_{C,F} \end{pmatrix} \end{array} \tag{4.2}$$

where each row represents a composite service $\hat{\mathcal{I}}i$, each column a functionality f_j, and each cell a component service of $\hat{\mathcal{I}}i$ referred in the matrix with the corresponding CM Instance $\mathcal{I}_{i,j}$=($cs_{i,j}, \mathcal{P}_{i,j}$,$p\text{-}eval_{i,j}$). Each service $\mathcal{I}_{i,j}$ is annotated with a sharing level $k \geq 1$, specifying the number of compositions $\hat{\mathcal{I}}i$ insisting on it. In the following, we denote the component service $\mathcal{I}_{i,j}$ selected as part of the composition $\hat{\mathcal{I}}i$ as $\hat{\mathcal{I}}i.\mathcal{I}_j$.

Example 4.2.3 *Figure 4.2(a) shows an example of deployment composition matrix D with 4 functionalities and 8 cloud services ($\mathcal{I}$), as follows:*

- *Functionality database (DB): mysql ($\mathcal{I}_1$) and posgresql ($\mathcal{I}_6$) are both certified for property confidentiality at different levels l.*
- *Functionality web application (WebApp): nginx ($\mathcal{I}_2$, $\mathcal{I}_4$, and $\mathcal{I}_7$) are certified for property vulnerability-free at different levels l. A level can refer to the severity of the CVSS score related to the CVE discovered on the target; the highest the level the lower the severity discovered.*

$$
\begin{array}{c} \\ \hat{\mathcal{I}}1 \\ \hat{\mathcal{I}}2 \\ \hat{\mathcal{I}}3 \end{array}
\begin{array}{c}
\begin{array}{cccc} DB & WebApp & Storage & Payment \end{array} \\
\begin{pmatrix}
\mathcal{I}_1 & \mathcal{I}_2 & - & \mathcal{I}_3 \\
\mathcal{I}_1 & \mathcal{I}_4 & - & \mathcal{I}_5 \\
\mathcal{I}_6 & \mathcal{I}_7 & \mathcal{I}_8 & \mathcal{I}_3
\end{pmatrix}
\end{array}
$$

$\mathcal{I}_1$ = {mysql.h-1, confidentiality, 1, ...}

$\mathcal{I}_2$ = {nginx.h-2, vulnerability-free, 4, ...}

$\mathcal{I}_3$ = {pay.remote, PCI-DSS compliance, 1, ...}

$\mathcal{I}_4$ = {nginx.h-4, vulnerability-free, 7, ...}

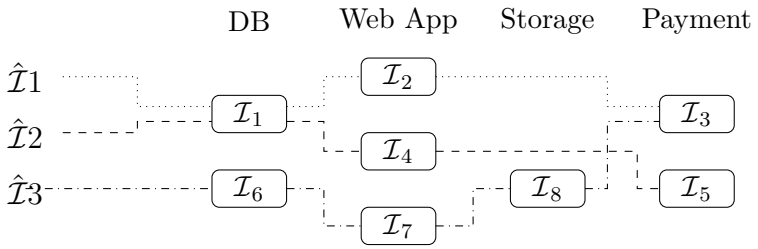
$\mathcal{I}_5$ = {ENGPay.remote, PCI-DSS compliance, 3, ...}

$\mathcal{I}_6$ = {psql.h-6, confidentiality, 0, ...}

$\mathcal{I}_7$ = {nginx.h-2, vulnerability-free, 10, ...}

$\mathcal{I}_8$ = {S3.h-2, e2e-confidentiality, 4, ...}

(a)

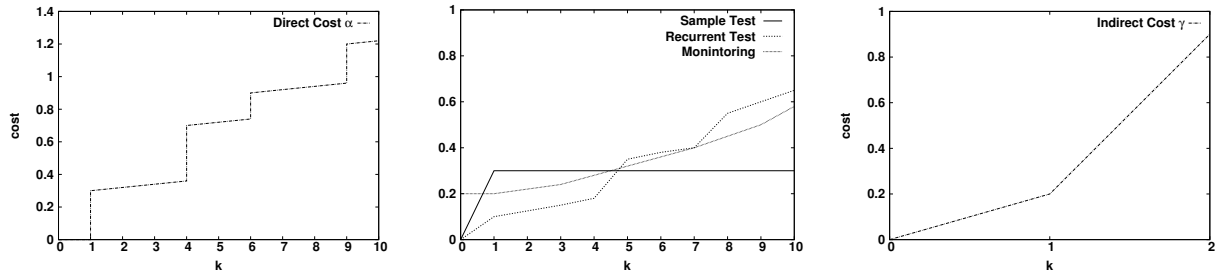


(b)

Fig. 4.2 An example of CP Status Matrix D with eight services and three compositions

- *Functionality storage (Storage): Amazon S3 ($\mathcal{I}_8$) are certified for property end-to-end confidentiality.*
- *Functionality payment (Payment): a remote payment service ($\mathcal{I}_3$) is certified for property PCI-DSS compliance level 1 and the ENGPay remote payment service ($\mathcal{I}_5$)is certified to provide PCI-DSS compliance level 3. Details about PCI-DSS compliance certification is available in [6].*

These services are composed in 3 composite services $\hat{\mathcal{I}}i$ (Figure 4.2(a)): i) *an e-commerce service $\hat{\mathcal{I}}1$ based on Database $\mathcal{I}_1$, WebApp $\mathcal{I}_2$, payment service $\mathcal{I}_3$;* ii) *a shared-economy app $\hat{\mathcal{I}}2$ based on Database $\mathcal{I}_1$, WebApp $\mathcal{I}_4$, and payment service $\mathcal{I}_5$;* iii) *an e-health service $\hat{\mathcal{I}}3$ based on Database $\mathcal{I}_6$, WebApp $\mathcal{I}_7$, storage $\mathcal{I}_8$, and payment service $\mathcal{I}_3$. Figure 4.2(b) shows a graphical overview of the composite services highlighting shared component services, that is, $\hat{\mathcal{I}}1$ and $\hat{\mathcal{I}}3$.*



(a) (b) (c)

Fig. 4.3 Cost functions

4.3 Cloud Service Provider Costs

The enrichment of traditional composition solutions with certification techniques evaluating non-functional properties of composite services introduces the need of rethinking the algorithms driving selection of component services. If, on one side, service selection has been already renewed to accomplish selection of services that prove a set of non-functional properties, on the other side, solutions to cost-based service selection need to depart from the assumption that costs are only due to service deployment and resource consumption [14]. The latter must consider costs introduced by certification processes, and by the need of keeping the composition continuously monitored and certified.

The management of a certified service introduces a cost on the cloud provider that can be evaluated using the Deployment Compositional Matrix and depends on three main factors: *direct (deployment) costs*, *indirect (certification) costs*, and *mismatch costs*. Each of these cost factors can be characterized using one of the four different cost behaviors identified by Horngren [124], and later used for cloud services by de Medeiros et al. [125]: *i) fixed costs*, a resource cost function that is completely independent from volume and time, indeed constant; *ii) variable costs,* a resource cost function that varies depending on volume or time, and is equal to zero for volume equal to zero; *iii) mixed costs,* a resource cost function that is the sum of a variable and a fixed cost function; and *iv) step costs,* a resource cost function that varies following different patterns.

4.3.1 Deployment Costs

$\alpha(\mathcal{I}.cs, \mathcal{I}.\mathcal{P}, \mathcal{I}.\text{k})$. They are defined by the cloud provider as the amount of resources to be allocated to a cloud service *cs* w.r.t. the certified properties $\mathcal{I}.\mathcal{P}$. They are usually estimated as a mixture of fixed deployment and variable usage costs [14]. Direct costs comprise servers, infrastructure, power, networks, and personnel costs []. They also consider the cost of orchestrating the composition, and managing service versioning and migration. An appropriate cost behavior can be a step function that depends on properties $\mathcal{I}.\mathcal{P}$ and the sharing level k, that is, the number of compositions insisting on a given service. Figure 4.3(a)

shows an example of direct costs; we observe a small cost increase between k=2 and k=4 due to power consumption, and a more substantial increase from k=5 when a vertical scaling of resources is required to satisfy all requests.

4.3.2 Certification Costs

$\beta(\mathcal{I}.p\text{-}eval, \mathcal{I}.k)$. They are defined by the cloud provider with the support of the certification authority as the amount of resources to be allocated to the certification infrastructure (Section 3.6) for continuous evaluation of service compositions. We note that, to execute evidence collection in *p-eval*, our certification infrastructure considers three types of collection activities (Figure 4.3(b)) having different costs.

- **Sample test**: one time evaluation of a specific part of the whole ToC. No need to keep evaluation resources allocated after the evaluation.
- **Recurrent test**: a scheduled, repeatable evaluation of a specific target; it is often part of a complex evaluation. The evaluation resources are permanently allocated to re-execute the evaluation multiple times.
- **Monitoring**: continuous and permanent evaluation on the target.

4.3.3 Mismatch Costs

$\gamma(\mathcal{T}.R, \mathcal{I}.\mathcal{P})$. They are defined by the certification authority, as inefficiency due to the distance between provided properties $\mathcal{I}.\mathcal{P}$ and required properties $\mathcal{T}.R$, with $\mathcal{T} \xrightarrow{I} \mathcal{I}$. Providing higher security properties means in general allocating more resources than needed (e.g., more computational effort for encryption with 128-bit key when 32-bit key was required). This loss of resources depends on the distance $Dist(\mathcal{I}.p, \mathcal{T}.r)$ for each $p \in \mathcal{P}$ and corresponding $r \in R$, according to property levels in Section 4.2. Figure 4.3(c) shows an example of mismatch cost function for a property/requirement distributed over three levels. We note that the function is not necessarily homogeneous over the number of property levels. For instance, it may observe an important boost for higher levels, because high security levels may require more hardware facilities.

4.3.4 Cost Profile

The CP behavior balances the contribution of direct, indirect, and mismatch costs to the computation of the total cost, while the total cost is the combination of each cost factor. We express the CP behavior as three cost profiles mapping to different CP strategies inspired by the deployment patterns in [126, 127].

- **Sharing profile** is typical of cloud providers that prefer to share resources, increasing the distance between requested and provided security properties (mismatch costs).

- **Fitting profile** is typical of cloud providers that prefer to achieve higher precision between requested and provided security properties, at the price of increasing the need of horizontal scaling. As a result, more component services are deployed precisely addressing on users' needs, decreasing the average sharing level (direct and indirect costs).
- **Average profile** where direct, indirect, and mismatch costs equally contribute to the total cost.

We note that a degeneration of the fitting profile where an ad hoc composition is deployed for each request is a good approximation of the actual strategy adopted by cloud providers.

4.4 Deployment Approaches

We propose a fuzzy-based cost evaluation approach, which evaluates the cost of composite services in Matrix D on the basis of cost factors and profiles in Section 4.3. Our solution aim to provide an accurate infrastructure and easy to tune approach for cost evaluation and profile setting.

4.4.1 Fuzzyfication

Fuzzy logic has been applied successfully in many fields including software cost estimation [128]. Let Y be the universe of discourse containing cost values y (i.e., cost factors in this Chapter). As customary, the membership degree of element y to a fuzzy set S is characterized by a membership function $\mu_S(y)$. A fuzzy set S in Y is denoted as follow.

$$S = \{(y, \mu_S(y)) | y \in Y)\} \tag{4.3}$$

where $\mu_S(y)$ is the membership function of the fuzzy set S. Let us then consider each cost function α, β, γ as a separate universe of discourse Y_α, Y_β, Y_γ. We define a standardized partition of each of them into different fuzzy sets. In the following, we only consider Y_α, because the same discussion applies to Y_β and Y_γ.

We define a standard fuzzy partition $\{S_{\alpha_l}, S_{\alpha_m}, S_{\alpha_h}\}$ of Y_α such that $\forall y_\alpha \in Y_\alpha, \sum_{j\in\{l,m,h\}} \mu_{S_{\alpha_j}}(y_\alpha)=1$. Each fuzzy set corresponds to a linguistic concept, that is, *LOW* for S_{α_l}, *MEDIUM* for S_{α_m} and *HIGH* for S_{α_h} costs. There are a number of different shapes for the membership functions related to each linguistic concept. In this work, we use R-function and L-function for LOW and HIGH membership functions and trapezoidal function for MEDIUM membership function. Given these linguistic variables mapping the concepts of LOW, MEDIUM, and HIGH costs for each cost factor, we define different sets of fuzzy rules. The rules, one for each profile, are used by the fuzzy inference system to infer the cost (Fc) introduced by each single component service $\hat{\mathcal{I}}t.\mathcal{I}_j$ at time t.

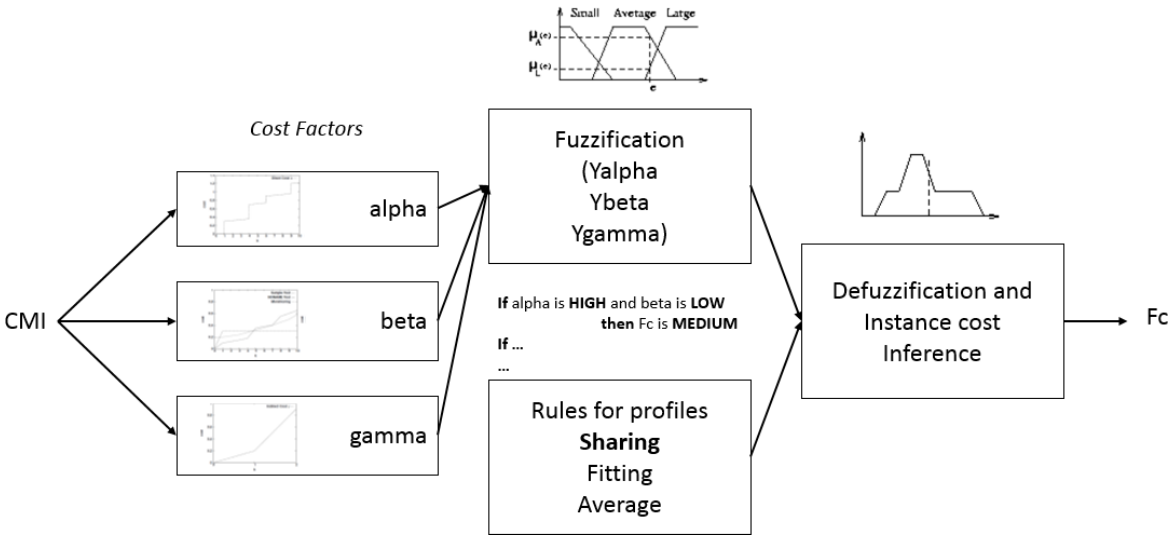


Fig. 4.4 Fuzzy cost inference for a given $\mathcal{I}$

Example 4.4.1 *Let us consider a component service $\hat{\mathcal{I}}i.\mathcal{I}_j$ ($\mathcal{I}$ for brevity) to be deployed at time t, following a sharing profile. Examples of fuzzy rules can be defined as follows:*
If $\alpha(\mathcal{I}.cs, \mathcal{I}.p, \mathcal{I}.k)$ *is HIGH and* $\beta(\mathcal{I}.\text{p-eval}, \mathcal{I}.k)$ *is not LOW and* $\gamma(\mathcal{T}.R, \mathcal{I}.\mathcal{P})$ *is LOW* ***then*** $Fc_t(\mathcal{I})$ *is HIGH.*
If $\alpha(\mathcal{I}.cs, \mathcal{I}.p, \mathcal{I}.k)$ *not HIGH and* $\beta(\mathcal{I}.\text{p-eval}, \mathcal{I}.k)$ *is not LOW and* $\gamma(\mathcal{T}.R, \mathcal{I}.\mathcal{P})$ *is LOW* ***then*** $Fc_t(\mathcal{I})$ *is MEDIUM.*
If $\alpha(\mathcal{I}.cs, \mathcal{I}.p, \mathcal{I}.k)$ *is LOW and* $\beta(\mathcal{I}.\text{p-eval}, \mathcal{I}.k)$ *is LOW and* $\gamma(\mathcal{T}.R, \mathcal{I}.\mathcal{P})$ *is LOW* ***then*** $Fc_t(\mathcal{I})$ *is LOW.*

Figure 4.4 shows the overview of our approach for fuzzy-based cost computation where, given a specific $\hat{\mathcal{I}}t.\mathcal{I}_j$ at time t, cost factors α, β, and γ are mapped to fuzzy domains (Θ). This mapping is based on standard partitioning into different membership functions, which are specific for each cost function.[2] Then, for each cost profile, a set of rules are defined and executed (Ξ) to infer the cost of each single $\hat{\mathcal{I}}t.\mathcal{I}_j$ ($\mathcal{I}$ in Equation 4.4 for brevity), using a defuzzification function (Ψ) as follows.

$$Fc_t(\mathcal{I}) = \Psi(\Xi(\Theta(\alpha(\mathcal{I}.cs, \mathcal{I}.p, \mathcal{I}.k), \beta(\mathcal{I}.p\text{-}eval, \mathcal{I}.k), \gamma(\mathcal{T}.R, \mathcal{I}.\mathcal{P})))) \tag{4.4}$$

Considering the CP status Matrix in Equation 4.2, total fuzzy cost TFc_t is the sum of the cost of each deployed composite service $\hat{\mathcal{I}}i \in \{\hat{\mathcal{I}}1, \ldots, \hat{\mathcal{I}}t\}$, with $t \leq |C|$. TFc_t can be formally

[2] Each CP tunes the membership functions of α and contributes to the tuning of the ones of β, while the membership functions of γ are defined by the CA.

expressed as

$$TFc_t = \sum_{\substack{\hat{\mathcal{I}}i \\ 1 \leq i \leq |C|}} \sum_{\substack{\hat{\mathcal{I}}i.\mathcal{I}_j \\ 1 \leq j \leq |\hat{\mathcal{I}}i|}} Fc_t(\hat{\mathcal{I}}i.\mathcal{I}_j) \tag{4.5}$$

where $\sum_{\substack{\hat{\mathcal{I}}i.\mathcal{I}_j \\ 1 \leq j \leq |\hat{\mathcal{I}}i|}} Fc_t(\hat{\mathcal{I}}i.\mathcal{I}_j)$ is the cost of a composite service $\hat{\mathcal{I}}$t and calculated as the sum of the costs of the corresponding component services $\hat{\mathcal{I}}t.\mathcal{I}_j$.

We note that total fuzzy cost is not the real cost incurred for a given deployment. It represents the cost perceived by a cloud provider according to the selected profile (i.e., sharing, fitting average) and the mixture of the different cost sources (i.e., direct, indirect, mismatch).

4.4.2 Heuristics

The aim of our solution is to find the best deployment $\{\hat{\mathcal{I}}1, \ldots, \hat{\mathcal{I}}n\}$ of composite services that *i)* satisfies a set $\{\hat{\mathcal{T}}1, \ldots, \hat{\mathcal{T}}n\}$ of composition requests, that is, guarantees $\hat{\mathcal{T}}i \xrightarrow{\hat{\mathcal{I}}} \hat{\mathcal{I}}i$, with i=1,...,n, *ii)* minimize the cost TFc_n (Equation 4.5) for the CP. We assume that a new composition request $\hat{\mathcal{T}}i$ is received every instant t, introducing a uniform time of arrival for composition requests. Finding the optimum deployment has however an exponential asymptotic behavior $\mathcal{O}\Big((|l| * F * t + t)^t\Big)$, in the worst case, with $|l|$ the number of property levels, F the number of functionalities, and t the number of composition request. Being t the dominating factor that varies over time, the exponential asymptotic behavior becomes $\mathcal{O}(t^t)$, which clearly does not fit the pseudo real-time requirement in our scenario. It is therefore necessary to design heuristic approaches for solving the problem in polynomial time, even for relatively large composite services.

Many heuristic approaches balancing efficiency and quality in terms of precision and recall can be used for minimizing TFc_t at time t, though not all of them are applicable in a cloud environment, where *i)* composition requests are consecutive, *ii)* the requests may need to be served quickly.

We propose two heuristic algorithms: *i)* heuristic *sliding window* that selects a cloud service deployment within a time-forwarding window w of composition requests $\hat{\mathcal{T}}i$ and *ii)* heuristic *sliding window with migration* that extends the heuristic sliding window with the possibility of migrating component services $\hat{\mathcal{I}}$i.$\mathcal{I}_j$.

Heuristic 1: Sliding Window

It is based on the idea of finding the best solution at time t using a time-forwarding window w. The heuristic selects the best deployment $\hat{\mathcal{I}}t$ at time t by evaluating a set of $|w|$ consecutive

requests $\hat{\mathcal{T}}i$, with $t \leq i \leq t+|w|$, received within window w. In other words, the selected $\hat{\mathcal{I}}$t represents the composite service contributing to the global optimum within window w.

At time t, the heuristic receives as input the CP status matrix D, which contains all deployed $\hat{\mathcal{I}}i$ with $1 \leq i \leq t-1$, all costs with related de-fuzzyfication functions, window size $|w|$, and the total fuzzy cost TFc_{t-1}.

Upon collecting the last $|w|$ requests $\hat{\mathcal{T}}t,\ldots,\hat{\mathcal{T}}t+|w|$, the heuristic calculates all possible candidate sets $\hat{\mathcal{I}}t,\ldots,\hat{\mathcal{I}}t+|w|$. For each candidate $\hat{\mathcal{I}}t,\ldots,\hat{\mathcal{I}}t+|w|$, our heuristic calculates the total fuzzy cost $TFc_{t+|w|}$ and chooses the one that entails the minimum increase of cost. The minimum increase of cost is calculated as the difference between the total fuzzy cost $TFc_{t+|w|}$ within window w and the current total fuzzy cost TFc_{t-1}. Both $TFc_{t+|w|}$ and TFc_{t-1} are calculated using Equation 4.4. Once the deployment $\hat{\mathcal{I}}t,\ldots,\hat{\mathcal{I}}t+|w|$ with minimum cost increase is selected, $\hat{\mathcal{I}}t$ is instantiated to satisfy request $\hat{\mathcal{T}}t$; the window is then shifted of one time interval and the process restarts to satisfy request $\hat{\mathcal{T}}t+1$ when a new request is received at time $\hat{\mathcal{T}}t+1+|w|$.

We note that $|w|$ must be chosen carefully to balance the quality of the retrieved solution and the performance/complexity of the overall heuristic. This decision is left to the CP based on its requirements or preferences. We also note that a degeneration of this approach with a sliding window of dimension $|w|$=1 yields to the greedy approach.

Heuristic 2: Sliding Window with Migration

It extends heuristic 1 with a better management of component deployment. Heuristic 2 supports service versioning and replacement (see Definition 4.2.5 and Definition 4.2.6), and in turn resource consolidation, for cost optimization. Migration in fact allows CP to modify its status matrix, moving to a new deployment scenario with lower costs. The global effect on the total cost, called *Migration Impact* (*mi*), is the difference between the total fuzzy cost TFc_t^{mi} after migration and the total fuzzy cost TFc_t before migration:

$$mi = TFc_t^{mi} - TFc_t \tag{4.6}$$

Migration impact mi<0 introduces a cost saving; migration impact $mi \geq 0$ introduces a cost increase.

A migration is triggered when a new composition request $\hat{\mathcal{T}}t$ is processed and results in the deployment of a new cloud service $\mathcal{I}_i$ first instantiated in $\hat{\mathcal{I}}t$ ($\hat{\mathcal{I}}t.\mathcal{I}_i$). For clarity, we describe our heuristic using compositions with a single functionality *f*, since every functionality is independent and therefore can be processed in parallel with the others. We then consider compositions $\hat{\mathcal{I}}$i that consist of a single component service $\mathcal{I}_i$ satisfying service request $\mathcal{T}_i$. The migration process is composed of two sequential phases and 4 steps as follows.

1. *Service migration*: this phase aims to optimize the cost of composite services $\hat{\mathcal{I}}$i in D at time $t-1$. In particular, it migrates component service $\hat{\mathcal{I}}$i.$\mathcal{I}_i$, such that $\hat{\mathcal{I}}i.\mathcal{I}_i \xrightarrow{\mathcal{I}}$

$\hat{\mathcal{I}}t.\mathcal{I}_t$, to the new cloud service $\hat{\mathcal{I}}t.\mathcal{I}_t$ deployed at time t (step 0) according to mi. Phase service migration starts with an ordering process, which introduces a migration priority among deployed services. Services $\hat{\mathcal{I}}i.\mathcal{I}_i$ are sorted in descending order according to function property distance $Dist(\hat{\mathcal{T}}i.\mathcal{T}_i.r, \hat{\mathcal{I}}i.\mathcal{I}_i.p)$ (see Equation 4.1), where $\hat{\mathcal{T}}i.\mathcal{T}_i.r$ is the property originally requested for composition request $\hat{\mathcal{T}}$i and $\hat{\mathcal{I}}i.\mathcal{I}_i.p$ is the property of the corresponding deployed composition $\hat{\mathcal{I}}$i (step 1). Once all services are sorted, for each $\hat{\mathcal{I}}i.\mathcal{I}_i$, the migration impact mi is calculated and, if $mi<0$, $\hat{\mathcal{I}}$i.$\mathcal{I}_i$ is migrated to $\hat{\mathcal{I}}t.\mathcal{I}_t$ (step 2).

2. *Resource consolidation*: this phase considers all component services $\hat{\mathcal{I}}$i.$\mathcal{I}_i$ in the CP status matrix D migrated during the previous phase. Since each service instance $\mathcal{I}_i$ is shared among different composite services, a migration changes the level of sharing of $\mathcal{I}_i$ and introduces the need of a consolidation process to optimize the total fuzzy cost TFc (step 3). Resource consolidation is a *binary join* operation between two services $\mathcal{I}_i$ and $\mathcal{I}_j$, with $\mathcal{I}_i.p<\mathcal{I}_j.p$, which migrates all service composition $\hat{\mathcal{I}}i$ that are deployed on $\mathcal{I}_i$ to $\mathcal{I}_j$, that is, $\hat{\mathcal{I}}i.\mathcal{I}_i$ is migrated to $\hat{\mathcal{I}}i.\mathcal{I}_j$. Among all possible pairs ($\mathcal{I}_i$,$\mathcal{I}_j$), heuristic 2 chooses the one that offers the best mi. Resource consolidation is recursively executed until no ($\mathcal{I}_i$,$\mathcal{I}_j$) offers a negative mi.

Example 4.4.2 *Let us consider a CSP offering compositions with a single functionality f and a property p with 3 levels. In the following, we describe the working of heuristic 2 as an extended version of heuristic 1.*

Step 0 – New request $\hat{\mathcal{T}}$t (Figure 4.5(a)). *A new request $\hat{\mathcal{T}}t$ at time t triggers the execution of heuristic 2. The status of the CSP is depicted in the status Matrix D in Figures 4.5(a), where each row represents the deployed composition $\hat{\mathcal{I}}i$, each column the deployed instance $\mathcal{I}_j$ offering functionality f and property p with level l, and each cell the request $\hat{\mathcal{T}}i.\mathcal{T}_j.r$ to be satisfied by the corresponding p of $\mathcal{I}_j$. For instance, in Figure 4.5(a), cloud service $\mathcal{I}_1$ offers a property p at level 1 ($p = (\hat{p}, 1)$) and is shared by composite services $\hat{\mathcal{I}}1$ and $\hat{\mathcal{I}}3$, whose templates $\hat{\mathcal{T}}1$ and $\hat{\mathcal{T}}3$ require property $\hat{\mathcal{T}}1.\mathcal{T}_1.r=(\hat{p}, 1)$ and property $\hat{\mathcal{T}}3.\mathcal{T}_1.r=(\hat{p}, 1)$, respectively. In the following, for simplicity, we consider the same $\hat{p}$ for both r and p, and then refer to levels r.l and p.l only. At time t, composition request $\hat{\mathcal{T}}8$ with r.2 is received. A composition instance $\hat{\mathcal{I}}8$ of $\hat{\mathcal{T}}8$ is deployed on a new cloud service $\mathcal{I}_4$ (denoted with a gray background in Figure 4.5(a)) offering p.2. We note that the result of step 0 is both the final result of heuristic 1 and the initialization step of heuristic 2.*

Step 1 – Service ordering (Figure 4.5(b)). *All component services $\hat{\mathcal{I}}i.\mathcal{I}_j$, with $1\leq j\leq 3$ and with $1\leq i\leq 7$ in our example, are then sorted in descending order by measuring distance $Dist(\hat{\mathcal{T}}i.\mathcal{T}_j.r, \mathcal{I}_4.p)$.*

Step 2 – Service migration (Figure 4.5(c)). *The corresponding migration impact mi in Equation 4.6 is then calculated for each of the $\hat{\mathcal{I}}i$ (denoted with a gray background in Figure 4.5(b))*

	$\mathcal{I}_1$ ($p.1$)	$\mathcal{I}_2$ ($p.3$)	$\mathcal{I}_3$ ($p.3$)	$\mathcal{I}_4$ ($p.2$)
$\hat{\mathcal{I}}1$	*r.1*			
$\hat{\mathcal{I}}2$		*r.3*		
$\hat{\mathcal{I}}3$	*r.1*			
$\hat{\mathcal{I}}4$		*r.1*		
$\hat{\mathcal{I}}5$			*r.3*	
$\hat{\mathcal{I}}6$			*r.2*	
$\hat{\mathcal{I}}7$		*r.2*		
$\hat{\mathcal{I}}8$				*r.2*

(a) Step 0

composition.cloud service	$\hat{\mathcal{I}}2.\mathcal{I}_2$	$\hat{\mathcal{I}}5.\mathcal{I}_3$	$\hat{\mathcal{I}}7.\mathcal{I}_2$	$\hat{\mathcal{I}}6.\mathcal{I}_3$	$\hat{\mathcal{I}}1.\mathcal{I}_1$	$\hat{\mathcal{I}}3.\mathcal{I}_1$	$\hat{\mathcal{I}}4.\mathcal{I}_2$
property distance	(+1)	(+1)	(0)	(0)	(-1)	(-1)	(-1)

(b) Step 1

order	*migration*	*migration impact*	*action*	$\mathcal{I}_4$*(p.2)*
1	$\hat{\mathcal{I}}7.\mathcal{I}_2 \xrightarrow{\hat{\mathcal{I}}} \hat{\mathcal{I}}7.\mathcal{I}_4$	*mi*=-2	migrate	$\hat{\mathcal{I}}t$,$\hat{\mathcal{I}}7$
2	$\hat{\mathcal{I}}6.\mathcal{I}_3 \xrightarrow{\hat{\mathcal{I}}} \hat{\mathcal{I}}6.\mathcal{I}_4$	*mi*=-1	migrate	$\hat{\mathcal{I}}t$,$\hat{\mathcal{I}}7$,$\hat{\mathcal{I}}6$
3	$\hat{\mathcal{I}}1.\mathcal{I}_1 \xrightarrow{\hat{\mathcal{I}}} \hat{\mathcal{I}}1.\mathcal{I}_4$	*mi*=+2	-	$\hat{\mathcal{I}}t$,$\hat{\mathcal{I}}7$,$\hat{\mathcal{I}}6$
4	$\hat{\mathcal{I}}3.\mathcal{I}_1 \xrightarrow{\hat{\mathcal{I}}} \hat{\mathcal{I}}3.\mathcal{I}_4$	*mi*=+3	-	$\hat{\mathcal{I}}t$,$\hat{\mathcal{I}}7$,$\hat{\mathcal{I}}6$
5	$\hat{\mathcal{I}}4.\mathcal{I}_2 \xrightarrow{\hat{\mathcal{I}}} \hat{\mathcal{I}}4.\mathcal{I}_4$	*mi*=-2	migrate	$\hat{\mathcal{I}}t$,$\hat{\mathcal{I}}7$,$\hat{\mathcal{I}}6$,$\hat{\mathcal{I}}4$

(c) Step 2

	$\mathcal{I}_1$ ($p.1$)	$\mathcal{I}_2$ ($p.3$)	$\mathcal{I}_3$ ($p.3$)	$\mathcal{I}_4$ ($p.2$)
$\hat{\mathcal{I}}1$	*r.1*			
$\hat{\mathcal{I}}2$		*r.3*		
$\hat{\mathcal{I}}3$	*r.1*			
$\hat{\mathcal{I}}4$				*r.1*
$\hat{\mathcal{I}}5$			*r.3*	
$\hat{\mathcal{I}}6$				*r.2*
$\hat{\mathcal{I}}7$				*r.2*
$\hat{\mathcal{I}}8$				*r.2*

(d) Step 3

	$\mathcal{I}_1$ ($p.1$)	$\mathcal{I}_2 \bigcup \mathcal{I}_3$ ($p.3$)	$\mathcal{I}_4$ ($p.2$)
$\hat{\mathcal{I}}1$	*r.1*		
$\hat{\mathcal{I}}2$		*r.3*	
$\hat{\mathcal{I}}3$	*r.1*		
$\hat{\mathcal{I}}4$			*r.1*
$\hat{\mathcal{I}}5$		*r.3*	
$\hat{\mathcal{I}}6$			*r.2*
$\hat{\mathcal{I}}7$			*r.2*
$\hat{\mathcal{I}}8$			*r.2*

(e) Step 3

Fig. 4.5 An example of heuristic 2 execution

showing a distance that is less or equal to zero (step 2). Figure 4.5(c) shows the results of our migration, that is, $\hat{\mathcal{I}}6$ is migrated from $\mathcal{I}_3$ to $\mathcal{I}_4$, and $\hat{\mathcal{I}}4$ and $\hat{\mathcal{I}}7$ are migrated from $\mathcal{I}_2$ to $\mathcal{I}_4$. We note that all these migrations are of type replacement as presented in Definition 4.2.6.

Step 3 – Resource consolidation (Figures 4.5(d) and 4.5(e)). *Upon phase service migration ends, phase* resource consolidation *is executed and considers all $\mathcal{I}_j$ such that at least one composite service $\hat{\mathcal{I}}i$ insisting on it has been migrated to $\mathcal{I}_4$. In our example, service instances $\mathcal{I}_2$ and $\mathcal{I}_3$ are candidates for the binary join (denoted with a light grey background in Figure 4.5(d)). Since the join between $\mathcal{I}_2$ and $\mathcal{I}_3$ has a negative mi=-2, resource consolidation is convenient and applied. Figure 4.5(e) finally shows the new CP status after the execution of heuristic 2, where the result of the join operation is denoted with a gray background. We note that all migrations due to consolidation are of type versioning as presented in Definition 4.2.5.*

4.5 Experimental Evaluation

We experimentally evaluated the performance and quality of our approach for cost-effective deployment of service compositions and the utility of our portable certification process. For conciseness, in this section, we report the most relevant results and findings of our experiments. Interest readers can access all results in Appendix C.

4.5.1 Experimental Setup

We considered a scenario where a cloud provider hosts three compositions as depicted in Figure 4.2. For simplicity but with no lack of generality, we focused on the payment functionality only, which is used in all compositions. Considering the entire composition does not give any additional insights on the soundness of the proposed approach and its performance/cost, since each functionality is treated independently and their cost summed up. CP offers two payment services, a standard payment service certified for property confidentiality, and ENGpay, a payment service offered by Engineering S.p.A. one of the biggest system integrator in Italy. ENGPay is offered with two levels of certification, a generic CIA (Confidentiality, Integrity, Authentication) certification and a more complex PCI-DSS compliance certification. We note that PCI-DSS compliance certification can be seen as an extension of the CIA certification. We then considered three different certification levels for property PCI-DSS compliance $\mathcal{P}_c$ from basic confidentiality ($\mathcal{P}_c$.level=1) to full PCI-DSS compliance ($\mathcal{P}_c$.level=3), via CIA ($\mathcal{P}_c$.level=2) .

We developed a *request simulator* that randomly generated requests for a payment service having a specific property level to the cloud provider, building 10 data sets of 300 consecutive random requests $\hat{\mathcal{T}}$, used to evaluate our deployment approach. For all data sets, we evaluated the deployment obtained using the sliding window and migration heuristics in Section 4.4.2 with sharing and fitting profiles in Section 4.3. To evaluate retrieved results we defined *i)* a set of evaluation metrics, *ii)* the fuzzy membership functions, and *iii)* the cost functions.

Evaluation metrics. We used four metrics to evaluate our approach.

Metric 1 measures the execution time needed to deploy composite services addressing composition requests.

Metric 2, called $\Gamma_t(TFc, TFc^{'})$, is the relative cost increment computed based on the area between two Total Fuzzy cost functions TFc and $TFc^{'}$ in the interval $[1, t]$. It is defined as follows:

$$\Gamma_t(TFc, TFc^{'}) = \frac{\sum_{i=1}^{t}(TFc_i - TFc_i^{'})}{\sum_{i=1}^{t} TFc_i} \tag{4.7}$$

where TFc_i and $TFc_i^{'}$ are the two Total Fuzzy cost functions evaluated at time i. We used Total Fuzzy cost to calculate Γ, since our goal here is to evaluate the overall cost increase and not the contribution of each cost factor (α, β, and γ).

Metric 3, called Δ_t, evaluates the cumulative number of portability events (versioning or replacement) occurred until a given time t. It provides a measure of how often a portability event and a consistency check are needed to support our dynamic composition certification, and in turns a measure of its utility.

Fuzzy membership functions. Our fuzzy system is based on membership functions and fuzzy rules that depend on the cost factors to be evaluated. We adopted the generalized bell-based memberships f for all cost factors as follows.

$$f(x; a, b, c) = \frac{1}{1 + |\frac{x-c}{a}|^{2b}} \tag{4.8}$$

where c is the center of the curve, a controls the width of the curve, and b controls the slope of the curve. To optimize the membership function definition we evaluated the distribution of costs to adjust the a, b, c parameters of each membership to the meaning of the corresponding linguistic variable. More precisely, we used a fuzzy c-mean approach to have an initial idea on the membership shapes using 100 requests from each of the 10 data sets. Given this shape we tuned the membership parameters to fit the fuzzy clusters with a *gbell* shape. We note that this process, as well as the cost function definition, is a tuning process that may depend on the cloud provider peculiarities. In general, the selected cost and membership functions are suitable for a generic cloud provider working with cloud service compositions, while the rule sets address the peculiarities of the profiles.

Cost functions. We used cost functions α, β, and γ in Figure 4.3 for the three property levels used in our experiments. We recall that cost function γ is defined using property levels and ranges from 0 to 2. α, β, and γ have been used to compare the cost retrieved by our heuristics (metric 2 and metric 3). Their definition is CP specific and should reflect the CP infrastructure costs. To fully evaluate our approach, we defined cost functions such that service migrations are triggered also with low numbers of composition requests.

We run our experiments on an Ubuntu server virtual machine with 22cores (physical machine Dell PowerEdge T620 equipped with 8 Xeon Octa Core 1.99 GHz), 16GB RAM, and 120GB HDD.

4.5.2 Performance evaluation

We evaluated the performance of our heuristics using *metric 1*, and cost factors α, β, and γ in Figure 4.3. Similar to the exhaustive algorithm, our heuristics have an exponential asymptotic behavior $\mathcal{O}\Big((|l| * F * |w| + t)^{|w|}\Big)$, with $|l|$ the number of property levels, F the

number of functionalities, $|w|$ the dimension of window w, and t the number of received requests. We note that, since $|w|$ is fixed a priori by our heuristics, the asymptotic behavior becomes polynomial as $\mathcal{O}(t^{|w|})$. Window w however makes our heuristics rapidly unusable given our assumption to serve request in pseudo real time (in the order of minutes), though their complexity is far lower than the one of the exhaustive algorithm, which is $\mathcal{O}(t^t)$.

Figure 4.6 compares the average execution time of the heuristics and exhaustive algorithm on the 10 data sets, varying window w from 1 to 7. Both sharing and fitting profiles show a similar performance trend just partially affected by optimizations based on branch cut. Heuristic 2 shows an additive cost increment with respect to heuristic 1 due to the migration and consolidation algorithms, which require sorting of compatible requests ($\mathcal{O}((t-1)^2)$ at time t, in the worst case). This additive factor is not anyways substantial, since it depends on the presence and amount of possible migrations (e.g., between $t = 70$ and $t = 80$). We note that the execution time of all algorithms is reported only for configurations requiring less than 3-minutes. Our results show that, as expected, the heuristics approximates polynomial execution time in the dimension of window w, which can be taken under control by selecting proper w. For instance, when w=7, execution time exceeds the 3 minute limit with a number $|\hat{\mathcal{T}}|$ of composition requests equal to 22 for heuristic 1 and 20 for heuristic 2; when w= 6, execution time exceeds the 3-minute limit with $|\hat{\mathcal{T}}|$= 155 for heuristic 1 and $|\hat{\mathcal{T}}|$= 143 for heuristic 2. The exhaustive algorithm shows the worst execution time, exceeding the 3-minute limit with $|\hat{\mathcal{T}}|$= 12. We note that the two heuristics have comparable performance dominated by w. We also note that the exhaustive behaves better in terms of performances in comparison to a given heuristics with window w for number of requests $|\hat{\mathcal{T}}| \leq w$ because the heuristics approach is forced to use a window size of w while exhaustive use the entire set of request. Therefore when the number of request is less that w it provide better or comparable performances.

4.5.3 Cost and Utility Evaluation

Performance evaluation in Section 4.5.2 showed the unmanageable complexity of the exhaustive approach, which required 21 minutes for 12 requests. We therefore compared the costs of our two heuristics on the 10 data sets and the utility of the portability underpinning them using *metric 2* and *metric 3* and cost factors α, β and γ.

We first evaluated the impact of window w on the relative cost increment (metric 2) with sharing and fitting profiles and on the entire set of requests $t = |\hat{\mathcal{T}}|$ and observed that the average relative cost increment ($\overline{\Gamma_t}$) across all the 10 data sets is negligible for fitting profile (i.e., less than 1%). Fitting profile in fact does not take significant advantages by looking forward in the incoming requests. In particular, there is an average cost degradation of 0.99% between window w=1 and window w=2, and in general, a cost degradation of 0.5% is observed between window w=1 and window w=5. For instance, a cost degradation has

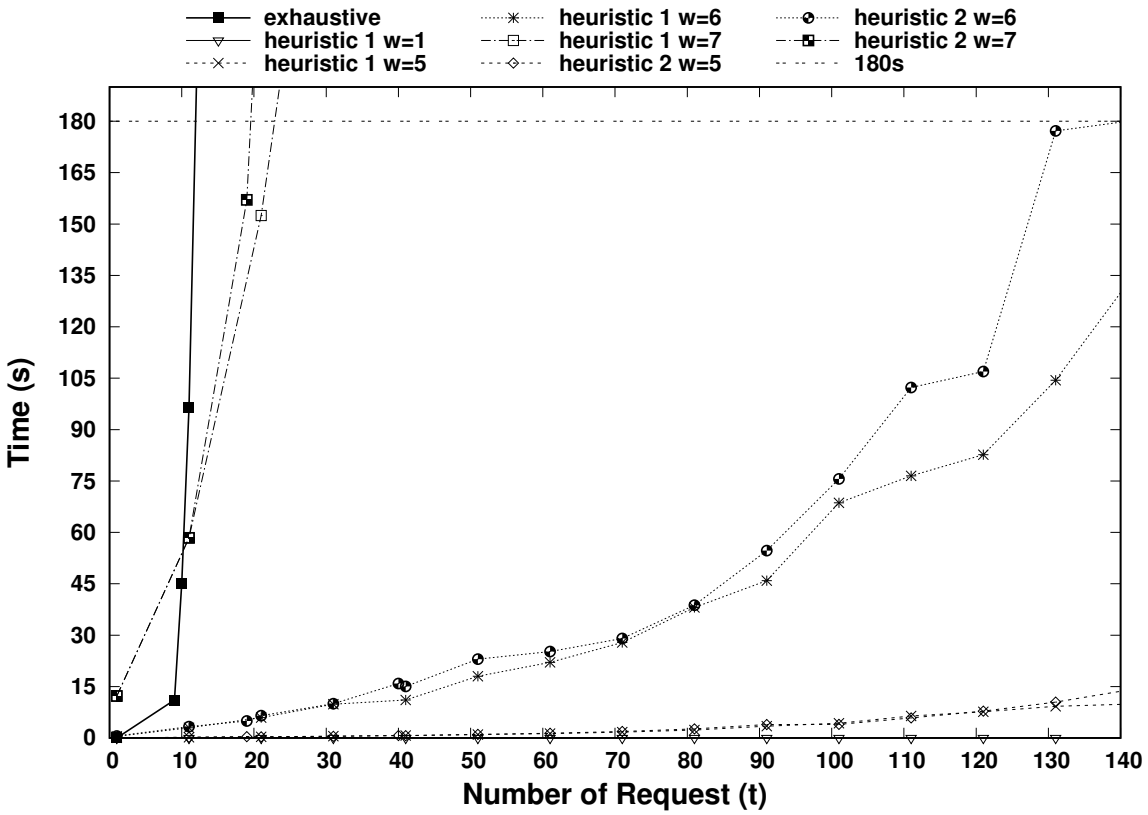


Fig. 4.6 Performance evaluation: heuristic 1 and heuristic 2 varying window size w.

been observed for data set 8 around t=290, where window w=5 suggests a deployment that increases the local cost with respect to the one suggested with a smaller window size; this degradation is the price we need to pay to have a lower cost after w=5 requests. [3]

Therefore if a CSP selects a fitting profile with the above cost functions and fuzzy memberships and rules, the window size is not impacting the cost optimization.

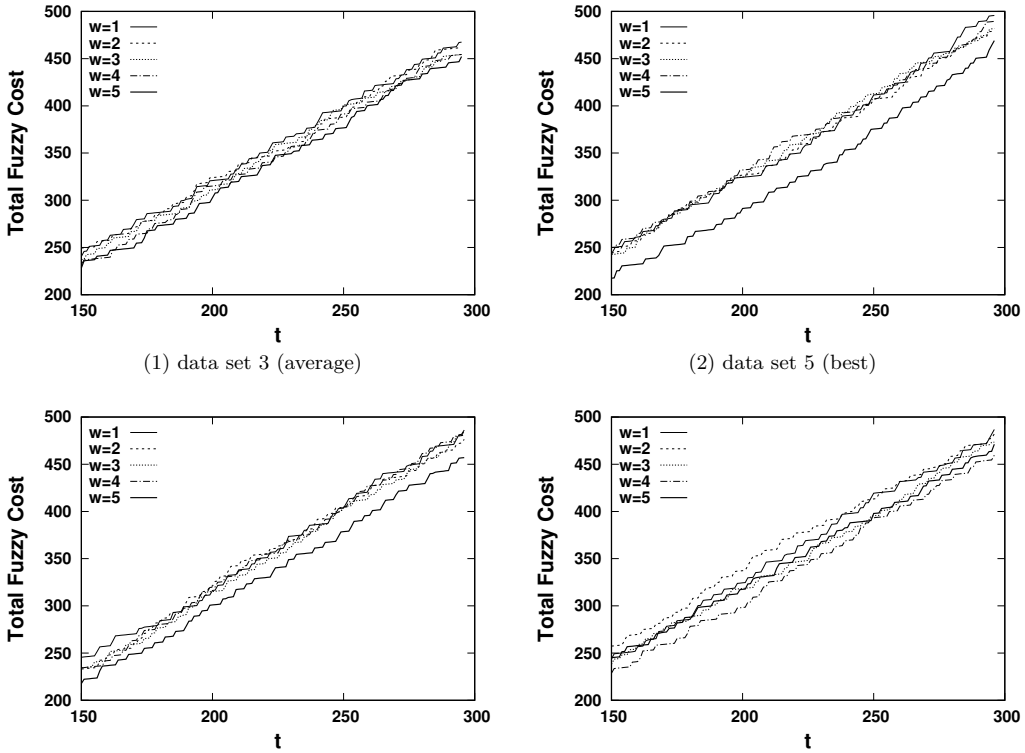
Our experiments with sharing profile instead show that the bigger w, the better (lower) the TFc. Figure 4.7(a) shows the average relative cost increment between deployments over our 10 data sets ($\overline{\Gamma_t}$) expressed in percentage comparing different windows size. The average cost increment is monotonic with the increment of the window size and is around 5.4% comparing window $w = 1$ with window w=5.

Figure 4.7(b1–b4) shows an excerpt of the Total Fuzzy cost TFc of heuristic 1 for 4 representative data sets (best, worst, average data sets) for sharing profiles, varying the window size from w=1 to w=5. We note that an increase in the cardinality of window w does not always result in a decrease of costs. Figure 4.7(b4) show a data set where heuristic 1 with w=4 has lower cost than the one with w=5. This mainly depends on the bias introduced by

[3] The choice of max window w=5 has been motivated by the need of taking performance under control (see Section 4.5.2).

	w1-w2	**w2-w3**	**w3-w4**	**w4-w5**	**w1-w5**
$\overline{\Gamma_t}$	-0.004%	-2.11%	-0.92%	-2.21%	$-5,37\%$

(a) *metric 2* (average % increase) $t = 295$



(1) data set 3 (average) (2) data set 5 (best)

(3) data set 8 (average) (4) data set 10 (worst)

(b) *metric 2* (TFc). We plot the most representative time frames (150–300) to improve figure readability

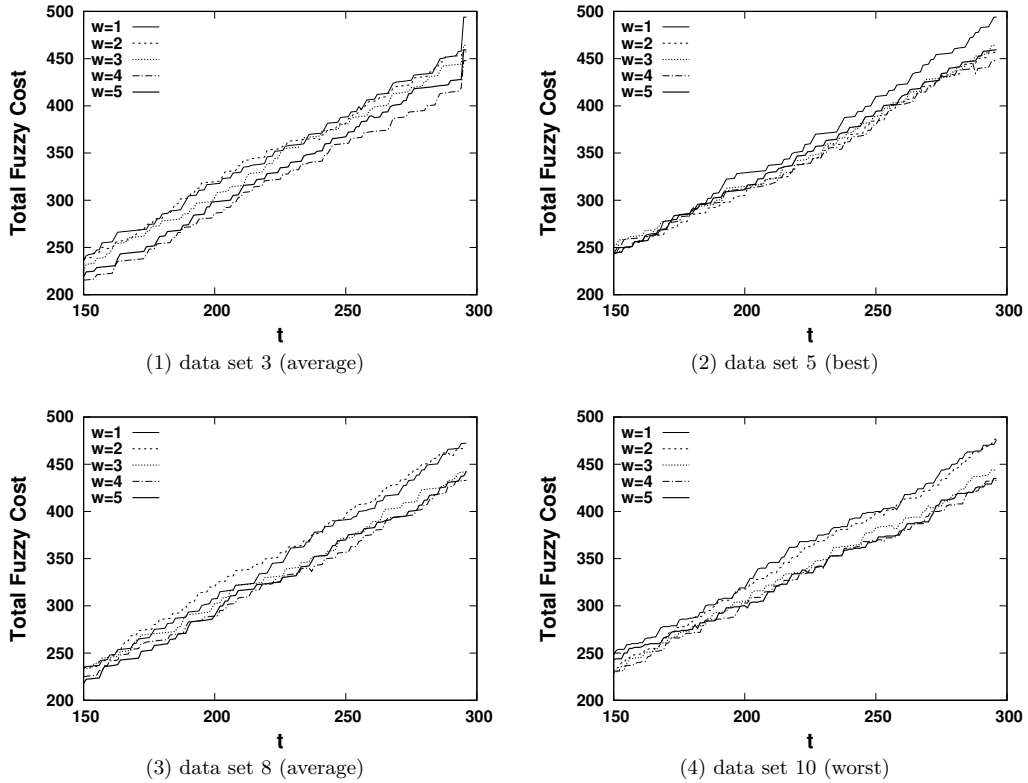
Fig. 4.7 Heuristic 1 cost evaluation for sharing profile varying window size w.

the random generation of the data sets and by the random selection of the best deployment when different candidate deployments have the same total fuzzy cost. This latter scenario, which may lead to a sub-optimal deployment drifting from the optimal total cost, is more probable at the beginning where there are more deployments with the same cost.

We note that heuristic 2 introduces migration between deployments to fix this issue and counteract drifting effects. Figure 4.8(a) shows the average relative cost increment between deployments ($\overline{\Gamma_t}$)of heuristic 2 with sharing profile and varying the window size w. We note that the behavior of heuristic 1 and heuristic 2 is similar, which is reasonable considering the refinement nature of heuristic 2. We also note that i) the drifting effects is reduced especially for bigger windows (greater that $w = 3$) and i) a reduced average cost improvement between $w = 4$ and $w = 5$ compared to the one of heuristic 1, meaning that heuristic 2 reduces the gap between this two window sizes. This effect is also clearly visible in Figure 4.8(b1–b4) that shows an excerpt of the Total Fuzzy cost TFc of heuristic 2 for the 4 representative data

	w1-w2	w2-w3	w3-w4	w4-w5	w1-w5
$\overline{\Gamma_t}$	-0.87%	-2.21%	-1.99%	-0.43%	-5.50%

(a) *metric 2* (average % increase) $t = 295$



(1) data set 3 (average) (2) data set 5 (best)

(3) data set 8 (average) (4) data set 10 (worst)

(b) *metric 2* (TFc). We plot the most representative time frames (150–300) to improve figure readability

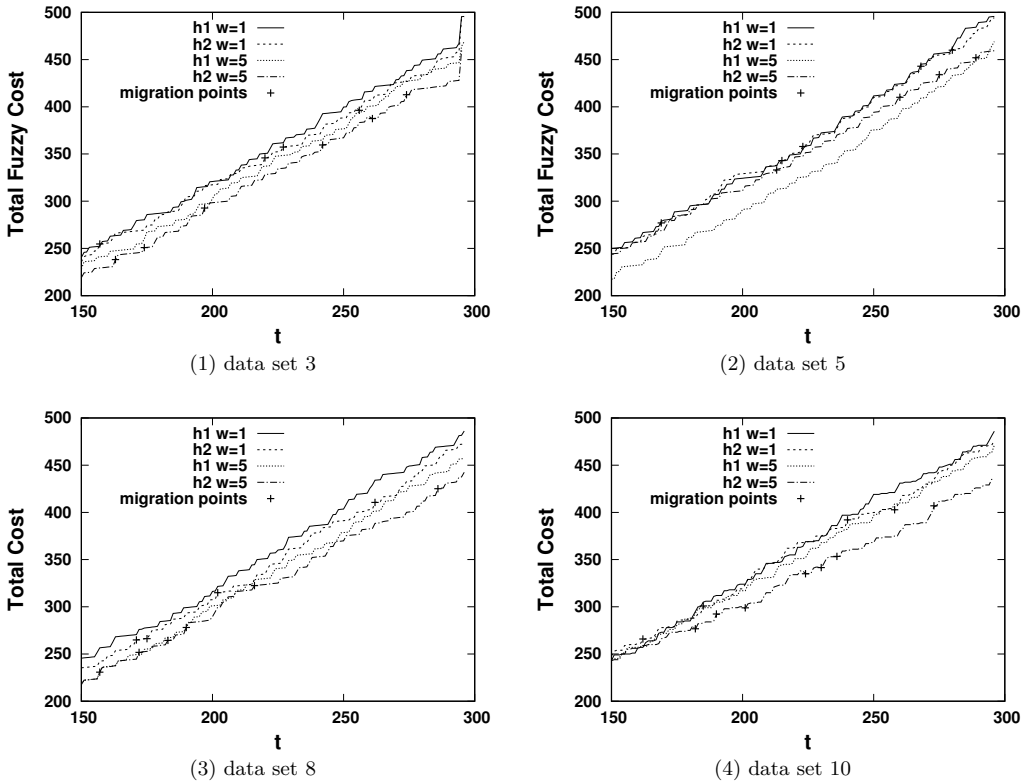
Fig. 4.8 Heuristic 2 cost evaluation for sharing profile varying window size w.

sets (best, worst, average data sets), where $w = 3$ and especially $w = 4$ has been improved making the behavior similar to $w = 5$ both in the best (b2) and the worst (b4) scenarios.

Heuristic 2 also generates a significant number of service migrations (metric 3). More in details, on average on the 10 data sets ($\overline{\Delta_t}$) for sharing profile and window $w = 5$ we counted 28 migrations. This non negligible number of migration events underlines the utility of certification portability supporting migrations (replacement) as well as new instantiations of the same services (versioning).

We therefore compared heuristic 1 and heuristic 2 with sharing profile only, using total fuzzy cost TFc, *metric 2* and *metric 3* and w=1 and w=5. The average relative cost difference ($\overline{\Gamma_t}$) between heuristic 1 and heuristic 2 with $w = 5$ is 1.09% showing that heuristic 2 improves heuristic 1.

Figure 4.9(b1–b4) shows the comparison of the total fuzzy costs TFc using sharing profile, for our 4 representative data sets (best, worst, average data sets). It also graphically shows the impact of migrations, which are marked in the Figure with "+". Figure 4.9(b3), at time



(1) data set 3 (2) data set 5

(3) data set 8 (4) data set 10

(b) *metric 2* (TFc). We plot the most representative time frames (150–300) to improve figure readability

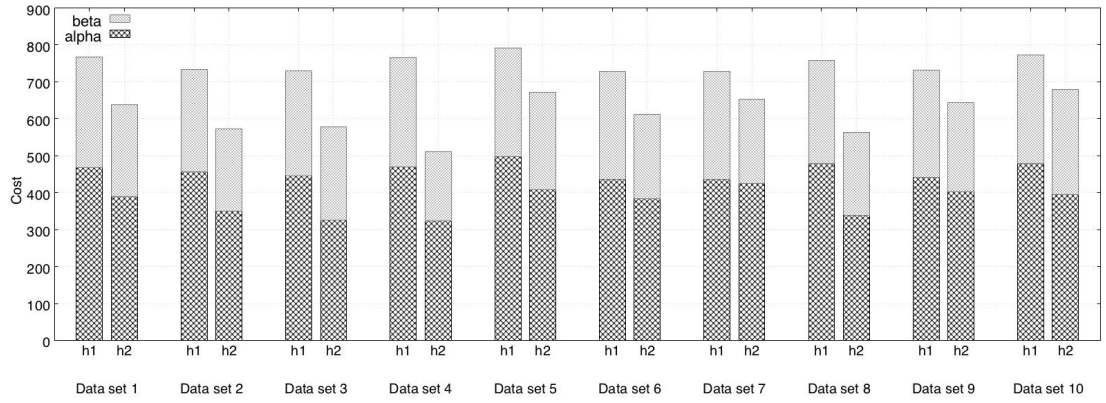
Fig. 4.9 Comparison between heuristics 1 and 2 with sharing profile, using w=1 and w=5. Migration events are marked with "+".

t=220, shows how migrations in heuristic 2 can lead to a better total cost, which lasts for a long period of time until the next migration at time t=280 occurs. Figure 4.9(b2), instead, shows how different consecutive migrations are triggered to re-calibrate the deployments and in turn the total cost (three migrations from t=250 to t=290).

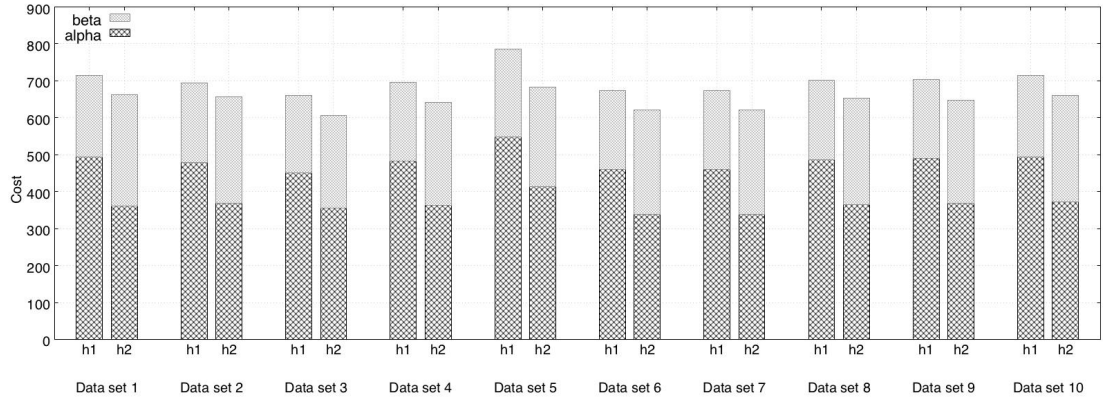
4.5.4 Discussion

Our approach enables CP to offer certified composition of services keeping internal cost under control. The fact that the certification process is portable is the ground for heuristic 2 and sharing profile that we proved (see Section 4.5.3) provide substantial benefits in terms costs.

As a general comment the cost evaluation presented in Section 4.5.3 being based on Total Fuzzy cost is affected by the "normalization" introduced by the defuzzification, and reflects a "perceived cost" more thatn a concrete cost. To provide an additional, more tangible quantitative analysis of the cost improvement, we present in the following a comparison based on the cost factors α and β.



(a)



(b)

Fig. 4.10 Comparing α, β total cost for the 10 data sets for sharing (a) and fitting (b) profiles using heuristic 1 window $w = 1$ (h1) and heuristic 2 $w = 5$ (h2) respectively.

Figure 4.10 shows a comparison in terms α and β for our heuristics for sharing and fitting profiles. In particular it compares i) heuristic 1 fitting profile $w = 1$ with heuristic 2 fitting profile $w = 1$ (a), and ii) heuristic 1 sharing profile $w = 1$ with heuristic 2 sharing profile $w = 5$. The improvement of using heuristic 2 instead of heuristic 1 on the real cost $\alpha + \beta$ are on average 18.5% for sharing profile and 8.2% for fitting profile respectively.

4.6 Chapter Summary

We proposed an approach to certification-based composition of cloud services. The proposed solution focuses on certification of non-functional properties of composite services and comparatively evaluates configurations of service compositions addressing clients' requirements. It also provides a cost evaluation methodology aimed to minimize the CP total costs of composition, identifying the best composition among possible alternative deployments. Three main cost factors have been considered in cost evaluation: *i)* traditional deployment costs, *ii)* certification infrastructure costs, and *iii)* mismatch costs. Our cost minimization approach has been experimentally evaluated according to three different cost profiles mapping different CP behaviors, comparing the average composition costs achieved by our incremental approach with the composition costs of the optimum approach.

Chapter 5

Cloud Service Assurance Framework

Lack of trust and transparency are among the main reasons hindering adoption of cloud computing. Users in fact can inspect neither their applications nor the treatment of their data, and have little or no guarantees about their security. Cloud security certification is a major assurance technique that has been proposed to increase cloud security, trust, and transparency. However, certification is a tedious, costly, and time-consuming process for the provider that wants to certify one of its services/applications. In this chapter, we propose a test-based security certification framework for the cloud implementing our certification process and a a cloud engineering methodology based on it, which supports providers in the design and development of ready-to-be-certified services/applications.

This chapter[1] proposes a test-based security certification framework for the cloud implementing our certification process described in Chapter 3. Addressing a more general problem, our framework is complementary to existing solutions for evidence collection (e.g., based on testing [17, 18] and monitoring [19, 20]) and can build on them to implement different certification processes, still maintaining the same deployment strategy. We claim that a framework implementing a security certification process for the cloud (Section 3.3) should support the following basic concepts. For the sake of clarity, requirements are prioritized according to modalities *MUST*, *SHOULD*, *MAY*.

[1] This chapter is based on the following publications:

- A Certification Framework for Cloud-based Services, Co-author: M. Anisetti, C.A. Ardagna, E. Damiani, published in Proc. of SAC 2016, April 2016, Pisa, Italy
- Modeling time, probability, and configuration constraints for continuous cloud service certification, Co-author: M. Anisetti, C.A. Ardagna, E. Damiani, N. El Ioini , published in Computers & Security (COSE) 72, 2018
- Moon Cloud: A Cloud Platform for ICT Security Governance, Co-author: M. Anisetti, C.A. Ardagna, E. Damiani, published in Proc. of IEEE GlobeCom 2018 , December 2018, Abu Dhaby, EAU

- *Distributed deployment*: the framework *MUST* be developed in a way that allows its deployment (all or in part) over the cloud.
- *Multi-layer and multi-target certification*: the framework *SHOULD* certify security properties of services and applications that can span different layers of the cloud stack.
- *Property-driven certification*: the framework *SHOULD* implement a certification process driven by the property to be certified for the system under certification.
- *Evidence-based certification*: the framework *MUST* support a certification process that provides a set of evidence proving a security property. The evidence *SHOULD* be collected in a standardized way (e.g., by agents and probes deployed within the system under certification), according to different collection mechanisms (e.g., testing, monitoring).
- *Incremental certification*: the framework *SHOULD* provide an incremental approach, meaning that evidence and corresponding certificates can be re-validated at runtime using an automatic and independent process. The security certificate awarded to a given system as the result of a certification process execution should undergo a continuous validation according to a specific life cycle.
- *Fully automatic configuration*: the framework *MAY* provide auto-configuration of agents and probes collecting evidence, thus supporting semi-automatic execution of (incremental) certification activities.
- *Extendible deployment*: the framework *MUST* be fully customizable and extendible to adapt to changing conditions.
- *Trusted implementation*: the framework and its activities *MUST* be trusted, increasing the confidence of the users in the correctness of the process and corresponding results.

Additional requirements should be addressed by the framework to support certified service engineering as follows.

- *Certification-aware cloud engineering*: the framework *SHOULD* provide a certification-aware cloud engineering approach that can be integrated with traditional development methodologies with limited effort.
- *Guided security mechanism development*: the framework *MUST* guide the development of those security mechanisms needed to certify a given property for a given system.
- *Step-by-step deployment*: the framework *MUST* drive the deployment of all components needed to support the execution of the certification process.

- *Certification process independence*: the framework *SHOULD* implement a methodology for certification-aware cloud engineering that is independent by the corresponding certification process.

In the remaining of the chapter, we first provide a detailed list of requirements that a framework for cloud service certification should meet. We provide then an overview of the developed framework describing its architecture, functionalities, the execution flows of an evaluation, and details on probe implementation and design. We finally introduce Moon Cloud, a spin-off of Universitá degli Studi di Milano, which provides an assurance platform based on the outcome of this thesis.

5.1 Requirements

In a scenario where the dynamics and rapid evolution of cloud services make assurance process over cloud services and security policies more complex and error prone, we elicits the following high-level security assurance requirements.

[R1] Security protection mechanisms. It requires suitable security protection mechanisms in place to protect the company assets. This is a mandatory requirement to achieve system security. Security mechanisms need to be selected from the market depending on the specific needs.

[R2] Holistic security. It requires a global, holistic, concise and clear view of the status of the security of the system. It is crucial to distribute the effort of the security specialists in improving the process and the policy; it departs from an evaluation that manually or, at most, semi-automatically inspects heterogeneous protection mechanisms.

[R3] Continuous monitoring of security protection effectiveness. It requires to continuously evaluate protection effectiveness for lowering the impact of human errors as well as process and organizational issues. Protection mechanisms in fact can be misconfigured over time due to the changing environment, coexistence of conflicting components, or errors; this scenario points to the need of continuously updating and verifying security mechanisms against vulnerabilities.

[R4] Efficient monitoring. Security monitoring is usually a runtime process that can interfere with the normal operation of the system under evaluation. A solution to continuous assurance monitoring should support an efficient process with low impact on the target system.

[R] Single point of management. It requires a single point of security management. It permits to keep the security policies under control with a holistic view ([R1]), avoiding proliferation of specific mechanism-related policies that can open the room to inconsistencies.

[R5] Asset inventory. It requires a clear view of the company assets to be protected, saving the sensitive data about them in a secure inventory. Asset inventory is the first requirement for every system. We note that the inventory must include all the assets, including service-level assets, as well as any external cloud/IoT asset.

[R6] Reliable evaluation. The reliability of the evaluation is crucial. It is important to evaluate the target during its normal usage (monitoring) and exercise it in specific critical situations (verification) when monitoring is not effective.

[R7] Intuitive protection status. High-level interpretation of each protection mechanism status is fundamental for *i)* avoiding the need of costly (technical) expertise, *ii)* supporting experts in recovering from security issues.

[R8] Policy-driven security. It requires to define a policy in a standard way to govern a process. Policy-driven security evaluation process means that the security evaluation activities reflect high-level policies supporting a more straightforward control and definition.

[R9] Custom policies. Security policies in [R8] should follow enterprise special needs. High level of customization is needed to tailor the security policies to the company needs.

[R10] Continuous compliance to standard. It requires to show compliance to one or more existing security standards to lower the risks of security incidents. Continuous compliance is achievable satisfying [R6] and orchestrating the activities following a policy as in [R8].

[R11] Internal security evaluation. All security mechanisms, including the one not exposed to the public, should be the targets of the security evaluation. Internal threats are in fact frequent and risky, and require fine-grained monitoring.

[R12] Fast reaction to security incident. The reaction to a security incident is often delayed by two factors: *i)* time to discovery of the security breach, *ii)* time to analyze the reasons of the breach.

[R13] Rapid and efficient a posteriori audit. In case of security incidents, an a posteriori audit process is needed to recover from the security breaches. Having a rapid and efficient audit process permits to re-establish the level of security.

[R14] Remediation. Being able to remediate, automatically or semi automatically, to security issues is fundamental for keeping the business continuity. If the security issues are not directly addressable, hints on how to remediate are needed.

5.2 Architecture

Our framework implements a master-slave architecture composed of two main modules: *Certification Manager* (CMF) and *Execution Manager* (EM) as showed in Figure 5.1. *Certification*

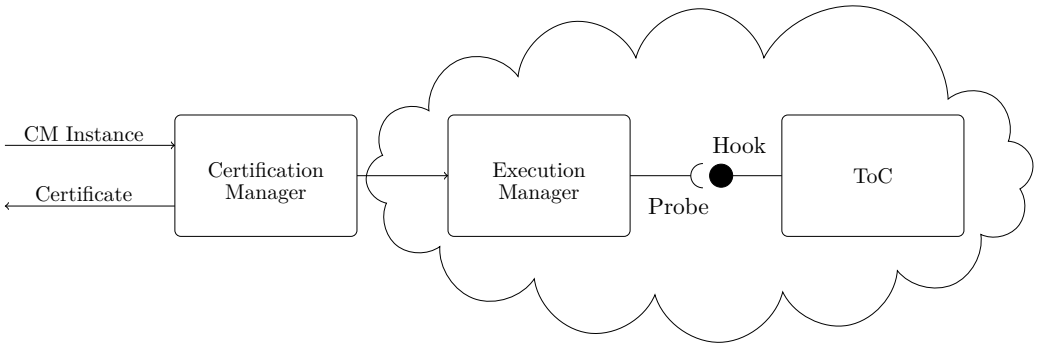


Fig. 5.1 A simplified view of the framework architecture

Manager – the master – is the owner of the certification process. It *i)* provides external API to manage (e.g., start, stop) a certification process, and to manage, retrieve, and modify corresponding CM Instances and certificates, *ii)* processes the CM Instance to extract the configurations and test cases needed for setting up the certification process and for initializing the slaves, and *iii)* manages the life cycle of existing certificates collecting aggregated results on testing activities. *Execution Manager* – the slave – is responsible for testing the ToC. It *i)* deploys the testing probes (see Section 5.2.4) and connects them to the hooks (i.e., the end points needed to access the target and execute test cases), *ii)* runs the test cases specified in the collectors, and *iii)* returns the results of the testing activities to TM.

Figure 5.1 shows the simple flows between CMF, EM, probes injecting test cases via hooks, and ToC. After a certification model instance is sent to TM, our certification process starts. The certification model Instance is parsed and split into pieces driving the activities of the running EM(s). EM sends back all evidence to CMF when available. The hooks are provided by the cloud/service provider and are generally protected by means of access control systems. CM aggregates the results returned by EM and either releases a new certificate, if possible, or manages the life cycle of an existing certificate.

Figure 5.2 shows a more detailed representation of the framework architecture and components, which is describe in the following.

5.2.1 Certification Manager

The *certification manager* (CMF) is the main component of the framework and orchestrates the whole certification and model verification processes, and a set of *execution clusters* aiming to collect evidence at the basis of certification and verification activities.

The certification manager stores all configurations and information needed to verify the correctness of the model (model structure, time constraints, probability constraints, configuration constraints, attack paths) and evaluates it once a sufficient amount of evidence is available. Certification manager is composed of the following modules:

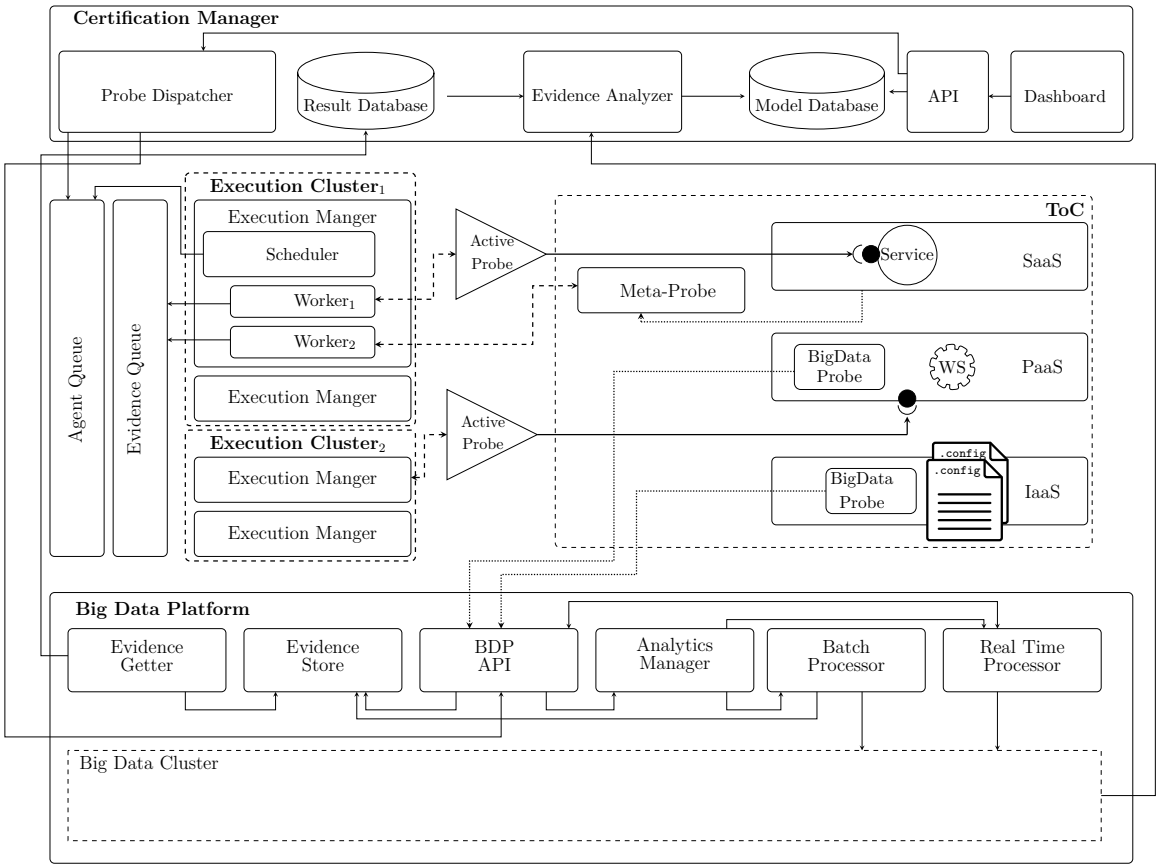


Fig. 5.2 A detailed view of the framework architecture

- *Dashboard*: a web application providing an interface to manage certification and model verification processes.
- *API*: RESTful api to communicate with the framework.
- *Model Database*: a database storing all configurations and rules for certification operations and model verification.
- *Probe Dispatcher*: a module implementing a communication channel between certification manager and execution clusters to require agent execution.
- *Result Database*: a time-series database automatically storing all evidence coming from execution clusters at the basis of certification and verification activities.
- *Evidence Analyzer*: a module analyzing the evidence in the result database and verifying the model correctness based on specified constraints.

Using a master-slave architecture, the certification manager may look as a bottleneck, but every single module was designed and developed to easily scale horizontally. Moreover every single module is deployed as a mircroservice and indeed scaled independently.

5.2.2 Execution Manager

Execution managers manage, deploy, and run agents on demand following requests from the certification manager on a full-duplex channel, where collected evidence are also exchanged. Communications are implemented using queues and the producer-consumer pattern. An execution cluster deploys multiple execution managers, each composed of the following modules:

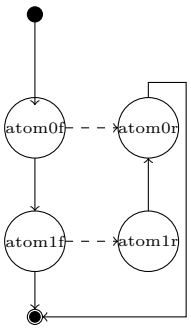
- *Scheduler*: a module attached to the agent queue and constantly waiting for messages from the certification manager. Upon a request arrival, it dispatches the task to a worker.
- *Worker*: a module directly connecting to agents in order to send them all configurations needed for their execution. Upon agent execution is completed, the resulting evidence is sent to the certification manager through the evidence queue.

We note that our design based on multiple EMs allows to scale the computational capacity of the system and, since the certification manager can choose the most appropriate execution cluster to exercise the ToC, to deploy and reach all cloud stack layers. In fact, it is possible to specify metadata for each cluster and then address the execution of test to specific EM. Moreover we design EM to be easily configurable, in fact, it doesn't require any special network configuration: this is possibile using as producer-consumer as communication patter that implies that only the queue must be reachable from both EM and CMF.

5.2.3 Big Data Platform

Due to the variety, velocity and volume produced by the certification process, the framework was extended to involve a component able to store and process Big Data. The Big Data Platform (BDP) collects data mainly from bigdata-probes that upload all retrieved data to the big data store, but it can also integrate evidence from Result Database. We note that bigdata-probes are described in details in Section 5.2.4. The Big Data Platform is composed by the following modules:

- Big Data Cluster: it is a cluster of bid data node able to execute real time and batch analytics.
- Evidence Getter: a module that get raw evidence from the Result Database periodically to extend data in the Evidence Store.
- Evidence Store: a database to keep big data evidence to process.
- BDP API: a RESTful interface that allows bigdata-probe to post data, Probe Dispatcher to instruct the Big Data Platform and in case of Real Time analytics to stream data to the Real Time Processor.



(a)

```
class SimpleEmptyProbe(Probe):

    def atom0f(self, inputs):
        #reading input
        port=self.testinstances["config"]["port"]
        ...
        return True
    def atom0r (self, inputs):
        ...
        return False

    def atom1f(self, inputs):
        #return evidence
        self.result.put("evidence-key","evidence-value")
        ...
        return True
    def atom1r (self, inputs):
        ...
        return False

#Definition of execution order
    def appendAtomics(self):
        self.appendAtomic(self.atom0f, self.atom0r)
        self.appendAtomic(self.atom1f, self.atom1r)
```

(b)

Fig. 5.3 Probe script in python. The probe is composed of two atom operations (*atom0*, *atom1*) with the corresponding rollback operations (*atom0r*, *atom1r*). Method *appendAtomics* specifies the order and matching of atom operations. *atom0* is executed first; *atom1* can access the results from *atom0* according to its definition.

- Analytics Manager: Based on the work of Ardagna et al. [129] the Analytics Manager is able to set up a big data analytics based on a given model.
- Batch Processor: a module connected to big data cluster able to deploy and run batch analytics.
- Real Time Processor: a module connected to big data cluster able to deploy and run real time analytics.

5.2.4 Probe

In Figure 5.2 is possible to identify three types of collecting modules, *active-probe*,*meta-probe* and *bigdata-probes* , working at different levels of granularity. The first module type, called *active-probe*, includes *testing* and *monitoring* functionalities. They directly exercise the ToC by executing test cases and/or by monitoring events, to the aim of evaluating the behavior (including attack scenarios) of the system under certification. Their main goal is to identify possible inconsistencies between the evidence retrieved in a laboratory environment and the

one retrieved in the production environment. The second module type, called *meta-probe*, is used for the verification of contextual (i.e., time, probability, configuration) constraints and system model correctness. They indirectly exercise the ToC by observing real and synthetic execution flows (traces) of the system under certification. Their main goal is to identify violations of time, probability, and configuration constraints, and inconsistencies of the execution flows. The *bigdata-probe* is usually installed as an agent inside the target and pushes continuously data to the *BDP API*. It doesn't provide any evaluation feature but it only accesses data, filters and structures them as expected by the BDP and then uploads them to the BDP where then they will be processed.

While bigdata-probes have a specific purpose, in some cases, active-probes and meta-probes can be used interchangeably. The choice depends on the specific scenario, and balance the quality of the retrieved evidence and the required level of access to the cloud infrastructure. Meta-probes connect to interfaces (hooks) provided by the cloud provider with limited access to the cloud backend; probes directly access the cloud backend and can manage part of it. As an example, both active-probes and meta-probes can be used to verify property confidentiality by encryption of a cloud storage. Active-probes require direct read/write access to the storage, while meta-probes only check whether the storage is configured to encrypt data without any access to the stored data. The use of meta-probes comes with some non-negligible advantages. First, they do not require the cloud providers to open their system to the outside and release sensitive data. Second, they verify support of a property without interfering with the normal execution of the system. Third, they introduce lower overhead. These advantages come at the price of a reduced quality of the evidence. In the following, we denote as structure, time, probability, and configuration (meta-)probes, (meta-)probes dealing with system model structure, time constraints, probability constraints, and configuration constraints, respectively.

Active-probes and meta-probes have the same structure (see Figure 5.3). They are composed of different atom operations that accept as input configuration parameters and results produced by previous operations, and returns as output evidence on the behavior of the system. An example of active-probe structure is showed in Figure 5.3. A probe is modeled as a State Transition System (STS) with two main flows of the same size (Figure 5.3(a)): *i)* the forward chain and *ii)* the rollback chain. The forward chain contains all the states that should be executed if there are no exceptions during the probe execution, otherwise the flow is redirected on the corresponding rollback state and continues on the rollback chain. This approach is designed to guarantee that the ToC can be always restored to the initial state. The example in Figure 5.3 shows a probe with 4 states : 2 forward states (i.e. atom0f and atom1f) and 2 rollback states (i.e. atom0r and atom1r). The order of execution and association between forward and rollback states are specified by the configurations in method *appendAtomics* (Figure 5.3(b)). We note that every state has as parameter *inputs*. This parameter is filled with the returned value by the previous state, indeed the first state

executed of the forward or rollback chain has *None* as *inputs*. The complete code of some example probes are available for interested readers in Appendix D.

For interested readers, we report and describe only simple example of probe that check the availability of a host by using ICMP-based method[130]. The code is shown in Figure 5.4. The probes is represented by the class *PingDriver* that provides three states:

- **init**: this state reads the string received as input (lines 3,4) that contains a series of hosts and trasform then in a list to pass to the next state.
- **ping**: ping receives in input the list of host to assess (lines 8,10). For each host it resolves the hostname if needed (line 14). If the host can't be resolve then it returns False (lines 15-18) otherwise it starts to ping the obtained IP (lines 20-25). if one of the host is not reachable the result of the state is False (29-32), otherwise it returns True and added as evidence IP and response time (lines34-37).
- **rollback1** This state, which is clearly a state of the rollback chain, returns False and an error as evidence.

The chain order is specified from lines 45 to line 48 in method appendAtomics. The forward chain involve first the init and then the ping methods. the rollback chain is composed by twice the rollback1 that store evidence only the first time is called through the trick at line 41.

5.3 Execution Flow

We report in this section two examples of the whole execution flow covering all operations from certification model instance $\mathcal{I}$ upload to certificate release. One example is based on the usage of active-probes while the other is based on the usage of bigdata-probes and the BDP.

Certification based on active-probes

The whole process depicted in Figure 5.11 shows the flow of an evaluation based on active probes, the flow can be split into 6 main steps:

1. **upload**: : User logs into the framework and then uploads, storse and requires the start of the CM instance $\mathcal{I}$ (operations a-b-c-d, Figure 5.5).
2. **start**: The *Probe Dispatcher* first parses the CM Instance $\mathcal{I}$and identifies the assurance tasks required by the certification process and then it uploads the assurance tasks on the specific *Communication Queue* (opeartions e,f,g, Figure 5.6).
3. **run**: EM gets the tasks from the queue and executes the probes against the target(operation h,i,j,k,l,m, Figure 5.7).

```
class PingDriver(Probe):
    def init(self, inputs=None):
        target = self.testinstances.get("config")
        host=target.get("host",None)
        assert host is not None
        hosts=host.split(",")
        return hosts
    def ping(self,inputs):
        return_value=True
        hosts=inputs
        name_ip={}
        result={}
        for h in hosts:
            ip=getAddr(h.strip())
            if ip is None:
                result[h] = {"status": "down"}
                result[h] = {"error" : "not reachable"}
                return_value=False
            else:
                name_ip[ip]=h
                result[h] = {"status":None, "IP":ip}
        hosts_toping=name_ip.keys()
        responses, no_response = multi_ping(hosts_toping, timeout=1, retry=2,
                                            ignore_lookup_errors=True)
        reachable=list(responses.keys())
        for r in result:
            if result[r].get("status",None) is None:
                value=responses.get(result[r].get("IP"))
                if value is None:
                    result[r]["status"]="down"
                    result[r]["errror"]="not reachable"
                    return_value=False
                else:
                    result[r]["status"]="up"
                    result[r]["time"]=\
                      float(("%0.3f"%responses.get(result[r].get("IP"))))
            self.result.put_value(r,result[r])
        return return_value

    def rollback1(self, inputs=None):
      if input is None:
            self.result.put_value("Error","Exception, contact admin")
        return False

    def appendAtomics(self):
        self.appendAtomic(self.init, self.rollback1)
        self.appendAtomic(self.ping, self.rollback1)
```

Fig. 5.4 Example of Probe script in python that checks if a host is reachable

4. **collect**: EM publishes all collected results on the *Evidence queue* and these are store in the *Result DB* (operations n,o,p,u, Figure 5.8).

5. **evaluate**: the *Evidence Analyzer* receive the evidence and based on the CM evaluation policy tags the single test as *pass* or *fail* and indeed tried to issue the certificate if the evidence all assurance task have been accomplished (operations q,r,s,v,x,w, Figure 5.9).

6. **release** The user once the certificate is finally issued can obtain it with the relative supporting evidence (operations t,y,z, Figure 5.10).

For a more detail description of the full sequence diagram in Figure5.11 we suggest to to follow Example 5.3.1.

Example 5.3.1 *Let's consider an Amazon S3 bucket*[2] *that must be certified for properties p=(*Confidentiality,*{*`ctx`=in-transit / at-rest*}). The certification model instance $\mathcal{I}$ will contains two security mechanisms, mapped to two test probes that will ensure that the given bucket is compliant with the general certification model template $\mathcal{T}$. This two mechanisms check that the bucket is encrypted and only accessible through https. The user, once logged (a), uploads the CMI through the API (b). $\mathcal{I}$ is validated and stored into the Model Database and a relative certificate in issuing state is then created for further operations (c). The API requests the start of the certification process to the Model Dispatcher (d) that once identified all test probes and relative test cases (e) (see Section 3.2) sends all the information to the designed execution cluster (f, g). The first Execution Manager with a free worker takes in charge of one of the two requested tests (h), deploying and running the probe (j) . Once the evidence is produced (k) then is sent back through the Evidence Queue(n). The arrival of new evidence on the Result DB (p) triggers the Evidence Analyzer (q) that perform the evaluation based on the available evidence (s). Since the $\mathcal{I}$required (r) two tests, the certificate can not be issued yet (s). Once another Worker finished the execution of test-2 (i, l, m, n) and evidence is stored into the Result Database (p) then the Evidence Analyzer can finally evaluate the $\mathcal{I}$(v,x) and release a Certificate (w). Since both tests were successful the user get from the api a positive certificate (y,z). We note that until the certificate is not issued then if a user require the certificate x (t) she will get back a certificate in issuing state.*

[2]https://aws.amazon.com/s3/

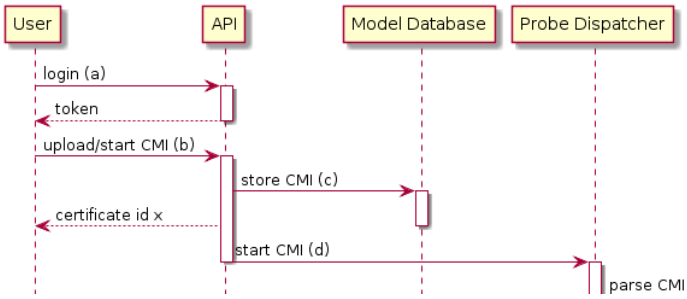


Fig. 5.5 Sequence diagram execution flow with active-probes, step *upload*

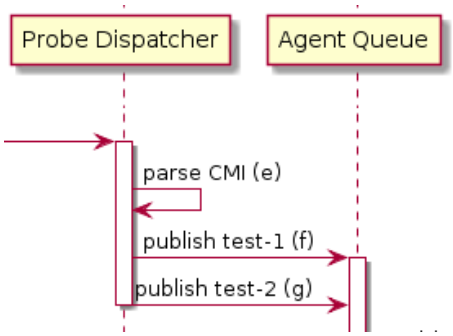


Fig. 5.6 Sequence diagram execution flow with active-probes, step *start*

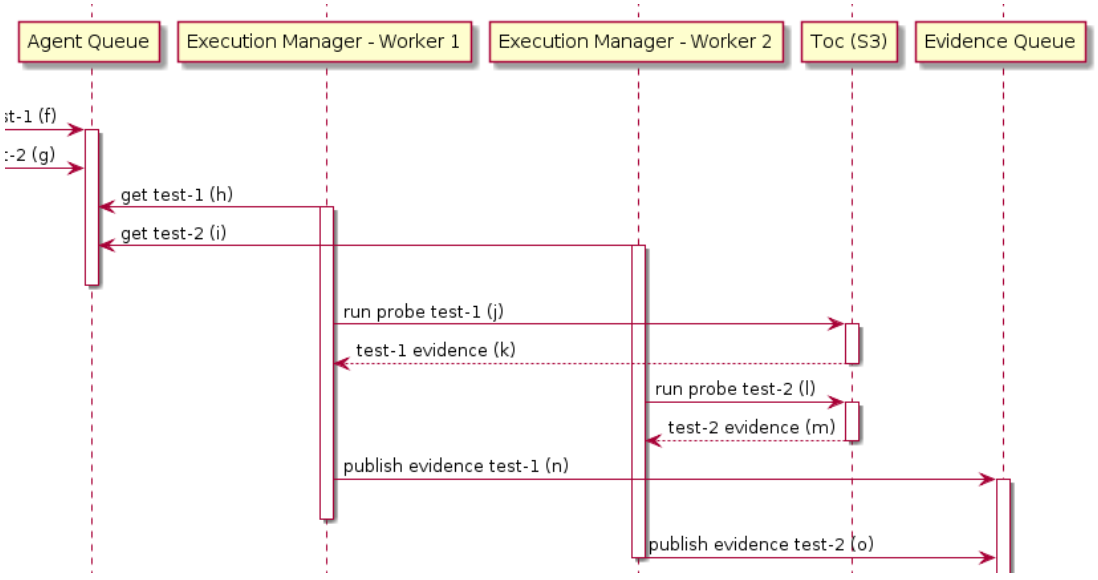


Fig. 5.7 Sequence diagram execution flow with active-probes, step *run*

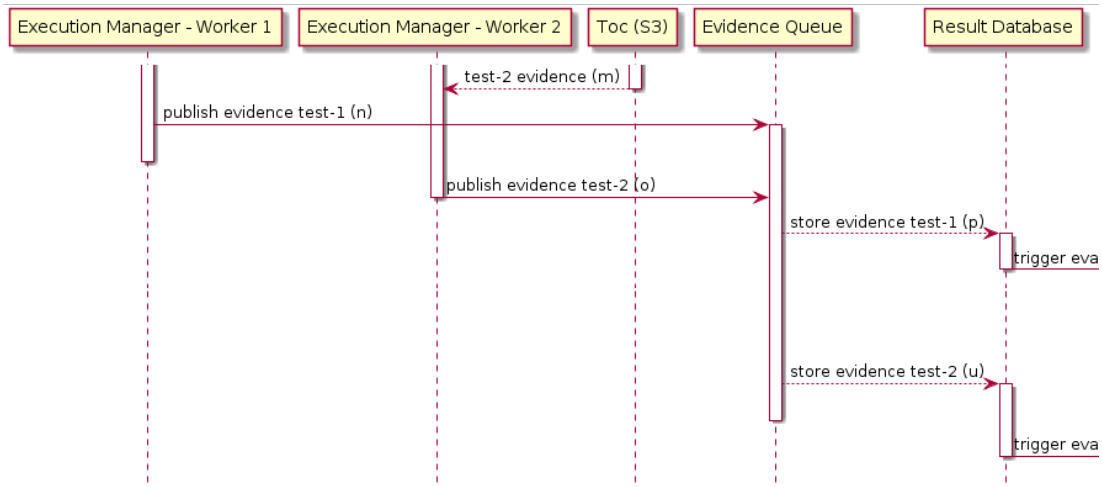


Fig. 5.8 Sequence diagram execution flow with active-probes, step *collect*

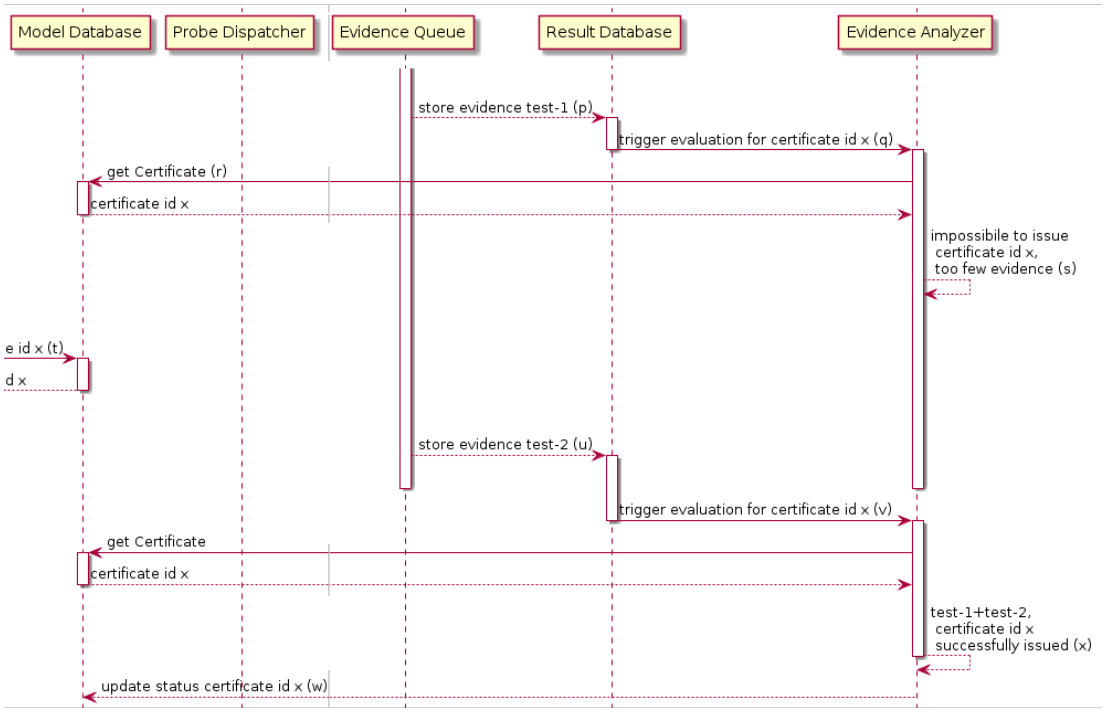


Fig. 5.9 Sequence diagram execution flow with active-probes, step *evaluate*

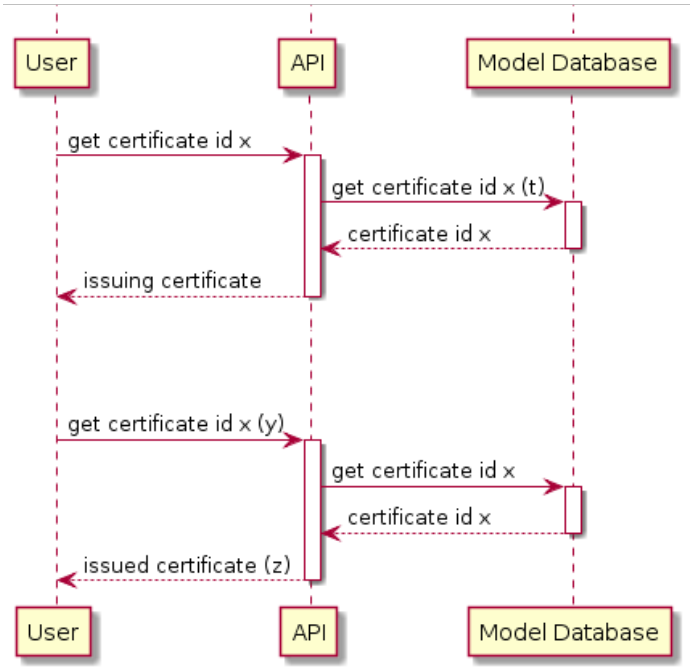


Fig. 5.10 Sequence diagram execution flow with active-probes, step *release*

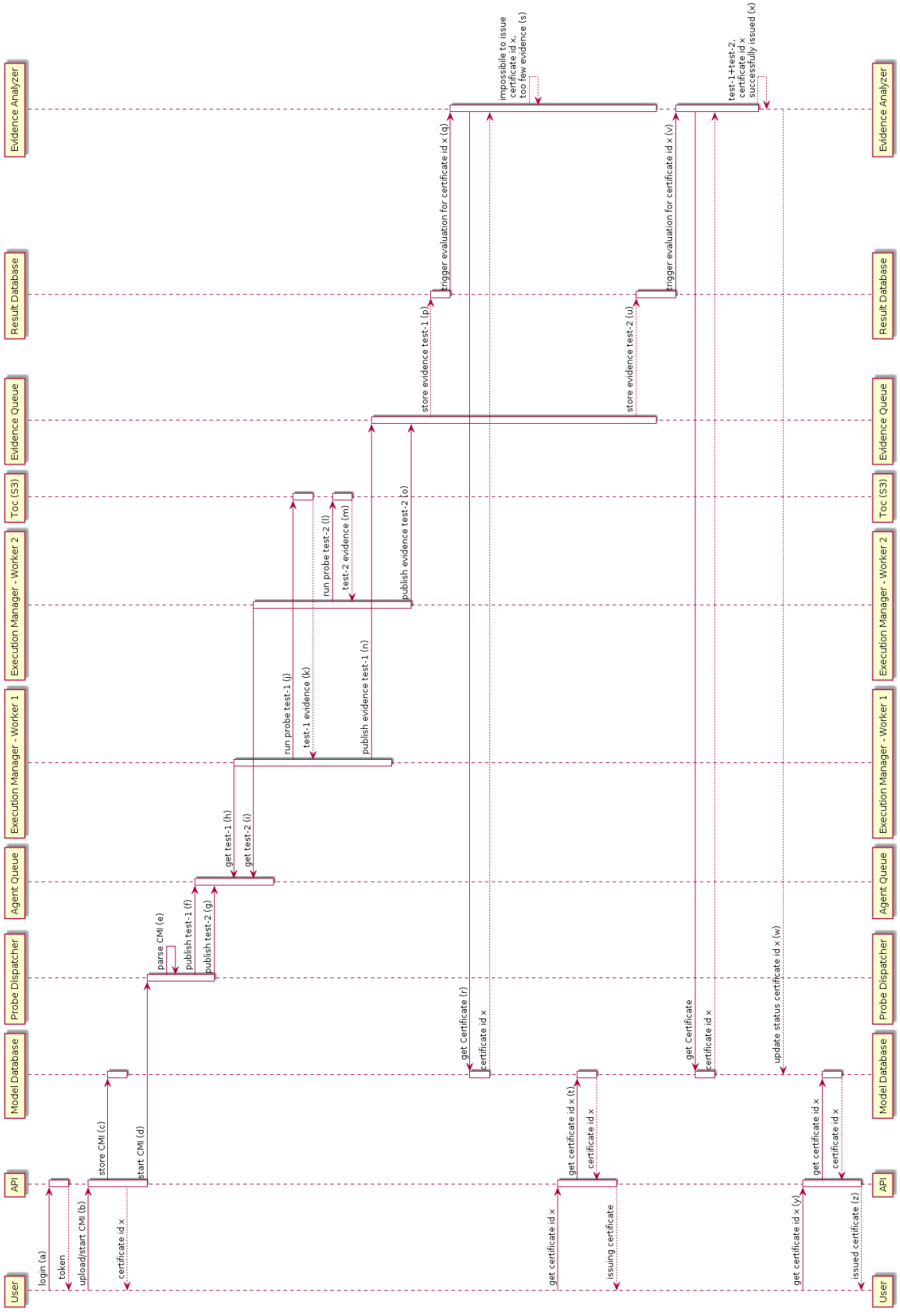


Fig. 5.11 Sequence Diagram of Example 5.3.1

Certification based on bigdata-probes

The execution flow of a certification operation using the BDP is depicted in Figure 5.18 and it can be summarized in the following operations[3]:

1. **upload**: User logs into the framework and then uploads, stores and requires the start of the CM instance $\mathcal{I}$(operations a-b-c-d, Figure 5.5)

2. **start**: the *Probe Dispatcher* parses the CMI and identified the bigdata-probes and the requested analytic. For every analytics the *Probe Dispatcher* instructs the BDP which configures the *Evidence Store*, the *Analytics Manager* and the *BDP API* to receive data from the specified bigdata-probes (opeartions e,f,g, Figure 5.12).

3. **run**: Bigdata-probes start uploading data into the BDP (operation h,i,l,m,n,o, Figure 5.13).

4. **setup analytics**: the *Analytics Manager* sets up and requires to run the *Batch Processor* that accesses the data in the evidence store and then deploy the required analytic in the *Big Data Cluster* (operations n,o,p,u, Figure 5.14).

5. **process**: the analytic is executed in the *Big Data Cluster* and the results sent to the *Evidence Analyzer* (operations, Figure 5.15).

6. **evaluation** the *Evidence Analyzer* receives the results of the big data process and, based on the CM evaluation policy, tries to issue the certificate (operations q,r,s,u,v,x,w, Figure 5.16)

7. **release** The user once the certificate is finally issued can obtain it with the relative supporting evidence (operations t,y,z, Figure 5.17)

For a more detail description of the full sequence diagram in Figure5.18 we suggest to to follow Example 5.3.2.

Example 5.3.2 *Let's consider a set of IoT devices that collect the city pollution that must be certified for properties p=(*Integrity*,{}). The certification model instance $\mathcal{I}$ will contains one security mechanisms, mapped to one batch analytic. The user, once logged (a), uploads the CMI through the API (b). $\mathcal{I}$ is validated and stored into the Model Database and a relative certificate in issuing state is then created for further operations (c). The API requests the start of the certification process to the Model Dispatcher (d) that once identified the bigdata-probes information and the requested analytics (e) sends both to the BDP API(f, g). The BDP API creates a dedicated store for the analytic in the* Evidence Store*, enables the reception of data from the specified bigdata-probe and pass the big data process model and the time to wake up*

[3]We note that some operation may coincide with the previous example.

to the Analytic Manager. *In the meanwhile the bigdata-probe starts uploading the data into the BDI API that then stores them into the* Evidence Store. Analytic Manager *automatically triggered by the wake up time sets the* Bath Processor. *The* Batch Processor *parse the model, get the data source from the* Evidence Store *and deploy the requested analytic on the* Big Data Cluster. *Once the analytic is done, the evidence is sent to the* Evidence Analyzer *and the Batch Processor is notified of the end. The* Evidence Analizer *can finally evaluate the $\mathcal{I}(v,x)$ and release a Certificate (w). Based on the evidence the certificate is issued and user can get from the api the certificate (y,z). We note that until the certificate is not issued then if a user require the certificate x (t) she will get back a certificate in issuing state.*

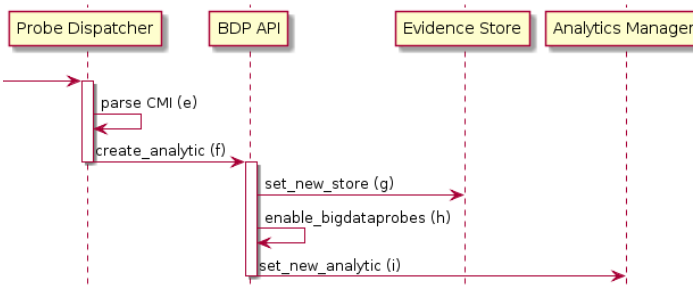


Fig. 5.12 Sequence diagram execution flow with bigdata-probe, step *start*

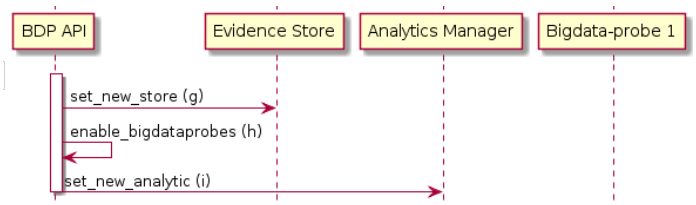


Fig. 5.13 Sequence diagram execution flow with bigdata-probe, step *run*

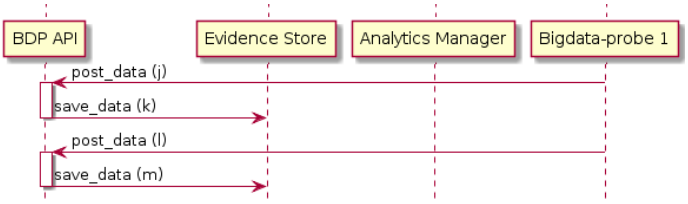


Fig. 5.14 Sequence diagram execution flow with bigdata-probe, step *setup analytics*

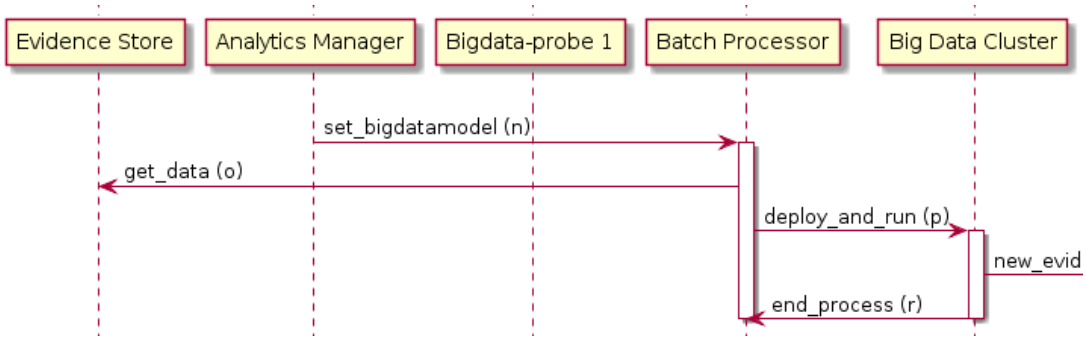


Fig. 5.15 Sequence diagram execution flow with bigdata-probe, step *process*

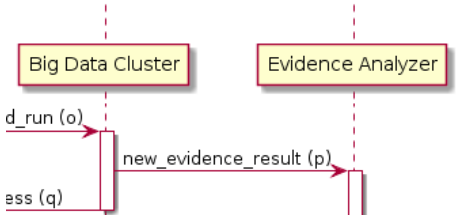


Fig. 5.16 Sequence diagram execution flow with bigdata-probe, step *evaluation*

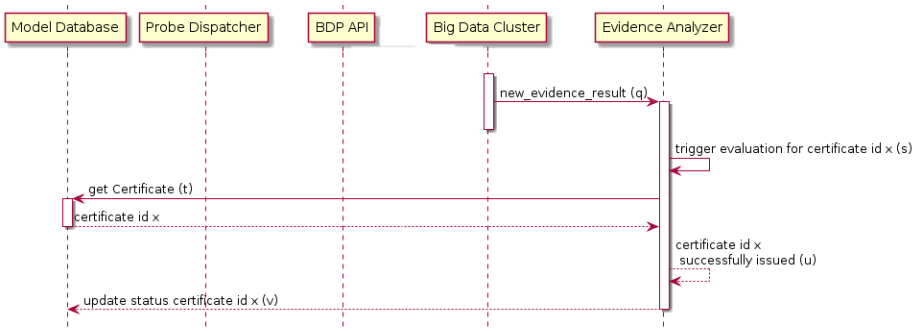


Fig. 5.17 Sequence diagram execution flow with bigdata-probe, step *releasee*

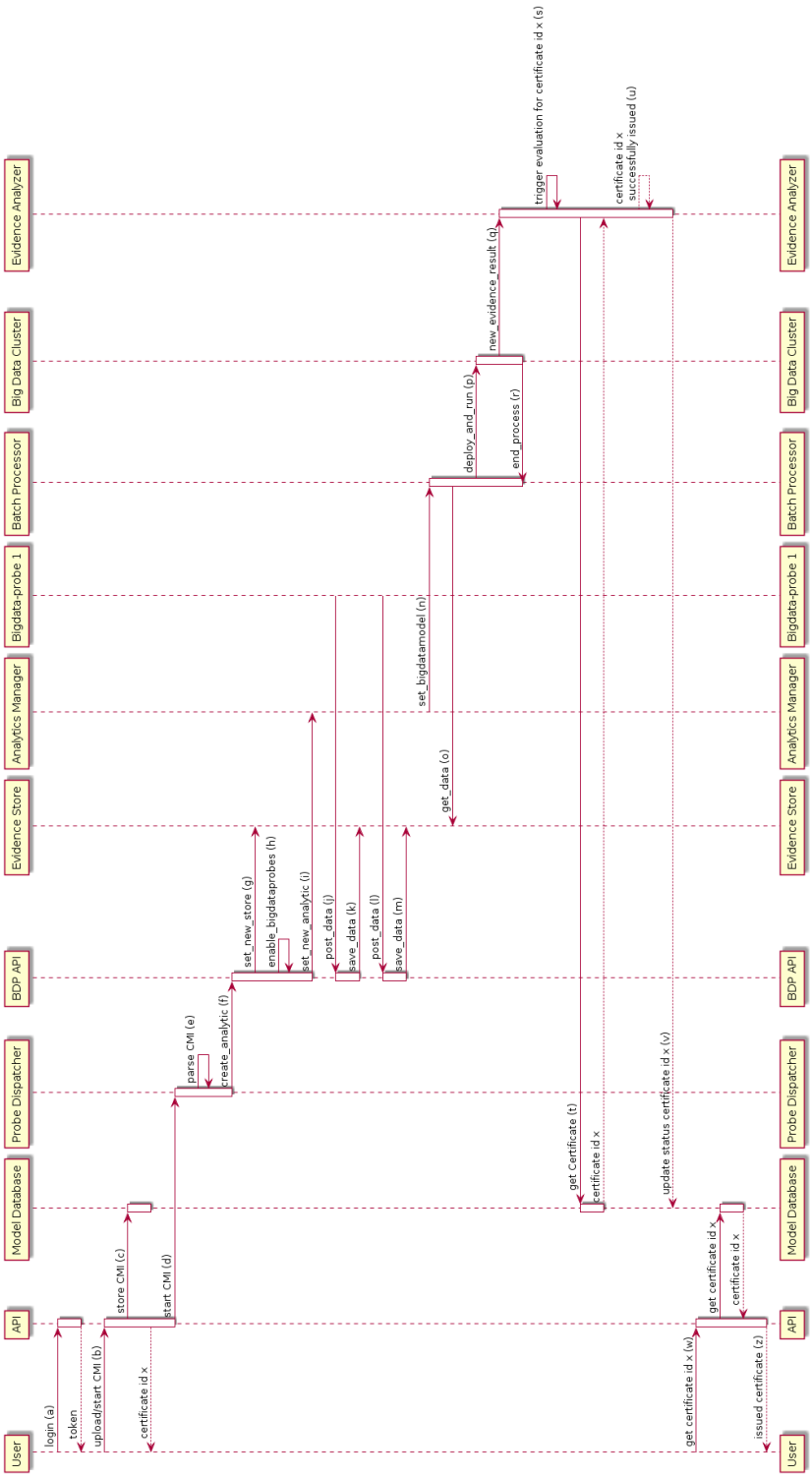


Fig. 5.18 Sequence Diagram of Example 5.3.2

5.4 Moon Cloud

Moon Cloud is a spin-off of Univestitá degli Studi di Milano. Moon Cloud is a platform for IT security governance and enables application owners to have a complete assurance evaluation and compliance assessment of their services during operations by means of continuous monitoring and testing. Moon Cloud mainly follows the architecture shown in Section 5.2 and implements the compliance process defined in Section 3.3 using monitoring/testing controls and a customizable security compliance evaluation fully compatible with probes described in Section 5.2.4. Moon Cloud is online and used by several customers in different scenarios (for more details see Chapter 6).

5.4.1 Mapping to Requirements

Moon Cloud covers requirements in Section 5.1 as follows.

- Moon Cloud offers a centralized cloud platform ([R4]), presenting a holistic view of the security status of a given system ([R1]). Moon Cloud platform aggregates the results of an assurance evaluation based on compliance into easy-to-understand metrics. These metrics are presented in an intuitive way ([R7]) and can be inspected by expert users together with each single evidence.
- Moon Cloud implements an evidence-based continuous assurance, implemented as a compliance process ([R2]) based on custom or standard policies ([R8][R9]), through a distributed and scalable set of probes. Probes are meant to capture specific evidence on both internal and external assets, which are aggregated for compliance evaluation.
- Moon Cloud is offered as a service – PaaS ([R3]), where the evaluation activities can be easily and efficiently configured to target the assets without the need of any additional human intervention. At the same time, it supports the design, implementation and deployment of custom policies ([R9]) and controls in a controlled environment, where the process of creating and building a control is semi-automatically carried out by the platform itself.
- Moon Cloud offers out of the box compliance to standards ([R10]), such as PCI-DSS, HIPAA.
- Moon Cloud permits to schedule automatic inspection, thanks to an asset inventory ([R5]) stored in a vault where all secrets are managed in a secure and confidential way.
- Moon Cloud evidence collection adopts both verification and monitoring techniques ([R6]).

- Moon Cloud evaluation engine can inspect the target from the inside ([R11]), managing internal threats.
- Moon Cloud permits fast reaction to security incidents ([R12]), thanks to a continuous evidence collection that supports efficient a posteriori audit ([R13]) and quick remediations.

5.4.2 Moon Cloud Architecture

Moon Cloud architecture is composed of an *assurance manager*[4] that orchestrates the whole evaluation via a set of *execution clusters*; each *execution cluster* manages and executes a set of probes collecting the evidence needed for the evaluation. The *execution cluster* can be deployed both externally, outside the target of evaluation, or internally (on premise), within the target of evaluation. All collection activities are executed by probes. Each probe is a python script provided as a single docker image, which is started when an evaluation is triggered and destroyed once the evaluation is done following the Function-as-a-Service (FaaS) pattern[5].

Upon accessing the Moon Cloud platform, the user can define its own security policies and evaluation activities as Boolean expressions of security controls and other predefined policies using the Moon Cloud dashboard. Once a policy is defined, the user can decide the scheduling time for policy verification, either one shot, discrete, continuous, and launch her evaluation. When a compliance evaluation against a policy starts, all controls are executed and the Boolean results collected. The Boolean results are then aggregated according to the policy and a true/false result returned to the user. The user can then access the results of the compliance evaluation through the dashboard at different levels of granularity: *i)* the overall system security status as a summary of all policy evaluation, *ii)* the result of each specific policy evaluation, and *iii)* the evidence supporting a specific policy evaluation.

When a compliance evaluation against the defined policy starts, all controls are executed and the Boolean results collected. The Boolean results are then aggregated according to the policy and a true/false result returned to the user. The user can then access the results of the compliance evaluation through the dashboard at different levels of granularity: *i)* the overall system security status (Figure 5.19(b)) as a summary of all policy evaluation, *ii)* the result of each specific policy evaluation (Figure 5.19(c)), and *iii)* the evidence supporting a specific policy evaluation (Figure 5.19(d)).

[4]We chosen *assurance manager* because it has been designed to be generic and support any assurance processes, including compliance, audit, certification.

[5]More details on FaaS at http://alexander.holbreich.org/serverless-manifesto/

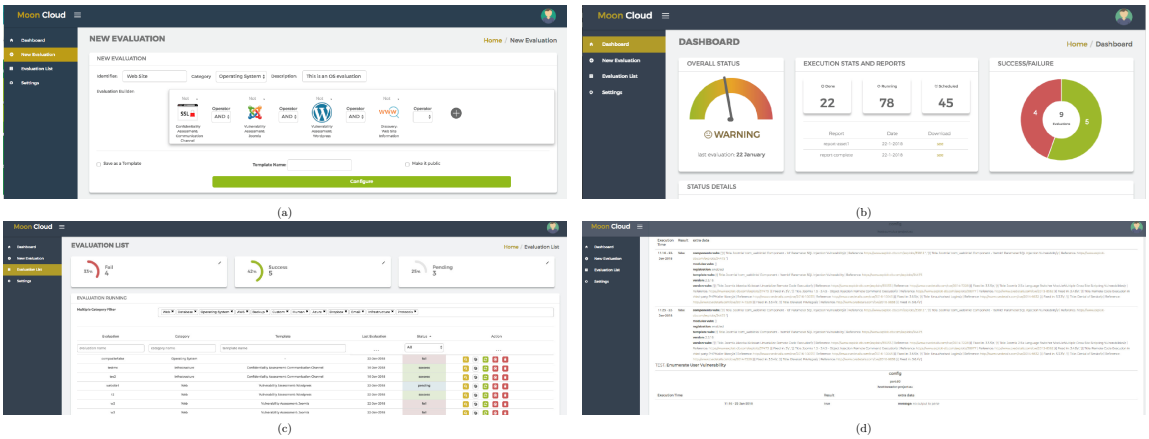


Fig. 5.19 Security governance with Moon Cloud

5.4.3 Implementation Notes

We describe some details about Moon Cloud and Probe implementation. API is a provided through NGINX server and is a web app all developed in AngularJS. All modules are microservices that provide a RESTful API implemented in Django RestFramework[6]. All documents and building blocks (i.e., CM Instances, agents, certificates) are stored in the *Model Database*, a Postgresql[7] database. Evidence instead, being time-series data, are stored in *Result Database* which is implemented by using InfluxDB[8]. Communications between CMF and EM are managed through a RabbitMQ queue with security features activated. The communication over RabbitMQ is secured by enabling the mutual authentication and the encryption of the channel. Using the *publish/subscriber* paradigm allows to scale horizontally over several Execution Managers as needed. All the platform is dockerized and indeed takes all advantages of containerization [131]. Execution Managers are python services that interacts with Docker hosts to deploy probes. A probe, in fact, is completely encapsulated in a docker container and run as a Function as a Service over a Swarm[9] Infrastructure. The Big Data Platform works with an Hadoop[10] YARN big data cluster using as process engine Spark[11]. The deployment of big dafa analytics is driven by Spring Cloud Data Flow [12] and the Evidence Store is provided through an H Base database.[13]

[6]http://www.django-rest-framework.org/
[7]https://www.postgresql.org
[8]https://www.influxdata.com/
[9]https://docs.docker.com/engine/swarm/
[10]https://hadoop.apache.org/
[11]http://spark.apache.org/
[12]https://cloud.spring.io/spring-cloud-dataflow/
[13]https://hbase.apache.org/

5.5 Chapter Summary

This chapter presented a certification framework that implements the security certification process for the cloud in Chapter 3 and 4. Our framework supports pre-deployment and production evaluation of cloud systems at different layers of the cloud stack. It also provides a certification-aware cloud engineering methodology that drives providers in the design and development of ready-to-be-certified systems. Our approach departs from traditional certification schemes, which are independent from the development process, to provide a certification scheme with reduced overheads. Moreover, this chapter presented Moon Cloud platform, a running solution for security governance and compliance assessment. Moon Cloud platform supports continuous verification, diagnostic, and monitoring of ICT system compliance against security policies. It provides an enhanced methodology that permits to correlate different and heterogeneous evidence to evaluate the status of the security of a given system.

Chapter 6

Application Scenarios

One of the main advantages of the cloud is the possibility to use and integrate managed services that completely run under the responsibility of third parties [132]. Relying on third party services can, on one side, boosts the development of new products and facilitate their maintenance by only focusing on the core business; on the other side, it comes at the cost of losing control over the whole product stack. Deploying an application on the cloud migth be straightforward thanks to the enabling technologies provided by CSP. When we come to security, it does not only depends on the application itself, but on the whole set of cloud services and infrastructure the application is built on. We define the cloud scope iceberg, shown in Figure 6.1, to highlight that from a point of view of the final users, the visible part only corresponds to the top of the iceberg, while it relies on the whole end-to-end chain. Indeed, to provide a secure system in the cloud, we need to consider all technologies, assets and actors involved.

In this Chapter[1], as a validation of what have been described in the previous chapters, we present two scenarios where security assurance can be applied to cloud services. The first is a practical scenario that focuses on the assurance evaluation of OpenStack, a major open source cloud infrastructure. To this aim, we define a security benchmark for OpenStack that is an instantiation and refinement of the CIS benchmark in [133] on the basis of the OpenStack security guidelines in [134]. This evaluation can be used in a composite certification to ensure that any applications built on OpenStack at least can rely on a secure infrastructure. The second scenario evaluates a web hosting service provided by Universitá degli Studi

[1]This chapter is based on the following publications:

- Toward Security and Performance Certification of OpenStack, Co-author: M. Anisetti, C.A. Ardagna, E. Damiani, R. Veca published in Proc. of IEEE CLOUD 2015 , June July 2015, New York City, NY, USA
- A Security Benchmark for OpenStack, Co-author: M. Anisetti, C.A. Ardagna, E. Damiani, published in Proc. of IEEE CLOUD 2017, June 2017, Honolulu, HI, USA
- Moon Cloud: A Cloud Platform for ICT Security Governance, Co-author: M. Anisetti, C.A. Ardagna, E. Damiani, published in Proc. of IEEE GlobeCom 2018 , December 2018, Abu Dhaby, EAU

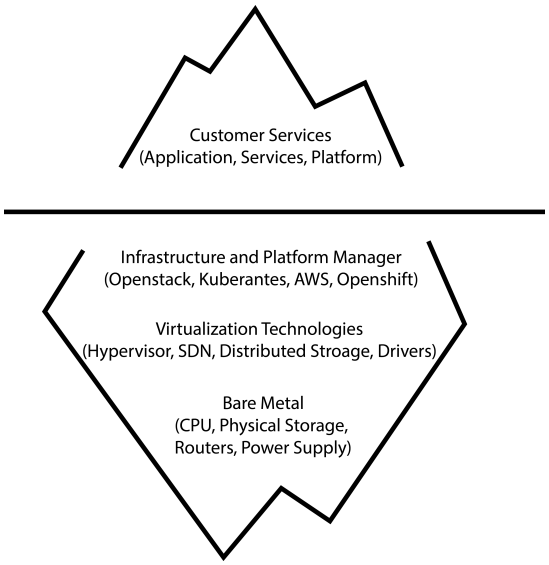


Fig. 6.1 Cloud Scope Iceberg

di Milano. In this case the evaluation covers the whole iceberg, from the websites down to the infrastructure and verifies it against the ICT security guidelines for Italian public administration provided by the "Agenzia per l'Italia Digitale" (AgID).

In both scenarios we use Moon Cloud, an implementation of the framework in Chapter 5, that provides a innovative B2B platform for large scale and continuous verification, diagnostic, and monitoring of ICT system compliance. For a more detailed description of Moon Cloud please refers to Section 5.4.

6.1 How to evaluate an IaaS Manager: OpenStack

In this Section, we focus on the evaluation of the security assurance of OpenStack, a major open source cloud infrastructure. We first define a security benchmark for OpenStack, inspired by Center for Internet Security (CIS) benchmark for cloud infrastructures. We then present a platform, called Moon Cloud, for cloud security assurance evaluation, showing an application of our benchmark and platform to the in-production OpenStack deployment of the University of Milan.

6.1.1 OpenStack

OpenStack is an open source IaaS solution providing functionalities for the management and monitoring of infrastructure resources. It is becoming a standard *de facto* due to its wide adoption by big IT Companies and is composed of the following set of core services.

- *Identity service (Keystone)*: it provides authentication and authorization to all services and supports different types of authentication.

- *Compute service (Nova)*: it provides Virtual Machine (VM) management through abstraction layers that support different hypervisors.
- *Object storage (Swift)*: it provides support for managing data in the cloud.
- *Block storage (Cinder)*: it provides persistent storage. It supports a full life cycle management for block storage, and access control and encryption functionalities.
- *Networking service (Neutron)*: it provides IP management, DNS, DHCP, load balancing, firewall policies, and VPN management.
- *Image service (Glance)*: it provides functionalities for disk image management.
- *Web dashboard (Horizon)*: it provides a dashboard used by users to interact with other OpenStack services.

6.1.2 Security Benchmark

A security benchmark is a set of (standard) recommendations against which the security strength of different systems can be compared. The recommendations are coupled with auditing activities specifying how to collect data for evaluating the recommendations. The result of a security benchmark evaluation is a score that represents the security strength of a specific product/service/deployment; the higher the score, the more secure the product/service/deployment.

The Center for Internet Security (CIS) provided a series of security benchmark for different solutions [133], ranging from applications such as Microsoft Word or MySQL, to Operating Systems such as Window Server or Ubuntu, and recently addressing cloud products such as AWS or Docker. Our analysis starts from the observation that OpenStack, though universally recognized as the most important open source cloud infrastructure (IaaS), does not come with a specific benchmark, but is just coupled with some high-level best practices proposed by OSSG (OpenStack Security Group). We therefore define a security benchmark for OpenStack as an instantiation and refinement of the generic CIS benchmark for IaaS systems in [135] on the basis of the OpenStack security guidelines in [134]. To this aim, we consider the OpenStack core services: *i) Keystone* is the identity service providing authentication and authorization for all users and services, *ii)* Nova is the compute service managing the life cycle of compute instances, *iii) Glance* is the image service providing a virtual machine disk image repository, *iv) Neutron* is the networking service providing network connectivity as a service, *v) Cinder* is the block storage offering persistent block storage, and *vi) Swift* is the object storage to store and retrieve arbitrary unstructured data objects.

Our security benchmark for OpenStack first maps the three profiles (*Virtual*, *Cloud*, *End User*) identified by the CIS benchmark [135] on OpenStack core services to address its peculiarities, including the concepts of shared responsibility and cloud layers, as follows.

Table 6.1 OpenStack Security Benchmark (OSB) addressing Hosts, OpenStack core services (Keystone, Nova, Glance, Neutron and Cinder, Horizon) and user configurations

#	Recommendation	Benchmark	Profile		
			Virtual	Cloud	User
R1	**Maintain Current Patch Levels**	CIS, OSB	✓	✓	✓
R2	**Create and Enforce Account and Password Management Policies**	CIS, OSB	✓	✓	×
R3	**Use a Central Directory for Authentication and Authorization**	CIS, OSB	✓	✓	✓
R4	**Configure Firewalls to Restrict Access**	CIS, OSB	×	✓	✓
R5	**Use TLS/SSL where Possible**	CIS, OSB	✓	✓	✓
R6	**Do Not Use Default Self-Signed Certificates**	CIS, OSB	✓	✓	✓
R7	**Configure Centralized Remote Logging**	CIS, OSB	✓	✓	✓
R8	**Maintain Time Synchronization Services**	CIS, OSB	✓	×	✓
R9	**Review and Minimize Hypervisor Attack Surface**	CIS, OSB	✓	×	×
R10	**Review and Minimize Virtual Machine Manager Attack Surface**	CIS, OSB	✓	✓	×
R11	**Use Templates to Deploy Virtual Machines**	CIS, OSB	×	✓	×
R12	**Disconnect unauthorized devices from Virtual Machines**	CIS, OSB	✓	×	×
R13	**Disable MAC Address Changes and Promiscuous Node on Guests**	CIS, OSB	✓	×	×
R14	**Ensure Network Isolation through VLANs**	CIS, OSB	✓	✓	×
R15	**Port Groups Should not be Configured to Reserved VLANs**	CIS, OSB	×	✓	×
R16	**Secure Access to Cloud Application Programming Interfaces**	CIS, OSB	✓	✓	×
R17	**Encrypt Data at Rest**	CIS, OSB	✓	✓	×
R18	**Establish Appropriate Resource Isolation**	CIS, OSB	✓	✓	✓
R19	**Evaluate Denial of Service Scenarios and Mitigations**	CIS, OSB	✓	✓	×
R20	**Do Not Use or Set Guest Customization Passwords**	CIS, OSB	✓	✓	×
R21	**Evaluate and Minimize Cloud Architecture Dependencies**	CIS, OSB	✓	✓	✓
–	**Align Infrastructure Security Controls with Tenant Requirements**	CIS	–	–	–
–	**Segment and Restrict User and Server Workload Communication**	CIS	–	–	–
–	**Restrict User-to-User Workload Communication**	CIS	–	–	–
–	**Deploy Anti-Malware Solution to End User Workloads**	CIS	–	–	–
–	**Audit Privileged Access to End User Workloads**	CIS	–	–	–
R22	**Audit sensible and configuration files**	OSB	✓	✓	×
R23	**Storage Reliability**	OSB	✓	✓	×
R24	**Data Remanence Avoidance**	OSB	×	✓	×

- *Virtual*: this profile pertains to all physical nodes where OpenStack services are installed, specifically addressing hardware configurations, Linux OSs, virtualization systems, and system service configurations.
- *Cloud*: this profile pertains to OpenStack services and add-ons. It involves OpenStack core services, the admin operations and configurations set.
- *User*: this profile pertains to OpenStack user usage. It addresses how users can secure their OpenStack projects, including VMs, virtual storage and network configurations.

We then instantiated the generic CIS benchmark recommendations and added some new recommendations on the OpenStack core services on the basis of the three defined profiles.

Table 6.1 presents an overview of our OpenStack Security Benchmark (OSB) presenting the security recommendations (field *Recommendation*), a comparison between recommendations supported by CIS and OSB (field *Benchmark*), and the corresponding profiles over which the properties insist (field *Profile*). Finally, starting from the best practices proposed by OSSG and following a strict architectural analysis of OpenStack, we defined for each recommendation the corresponding security control(s).

[R1] Patch Levels. Continuously check the version of installed software including OpenStack services (*Virtual*, *Cloud*, *User*)

[R2] Create and Enforce Account and Password Management Policies.

- Enable PAM, LDAP, or similar authentication systems for every host and allow certificate-based authentication only. Use certificate rotation and minimize root accesses. Disactivate users after long inactivity (*Virtual*)
- Use password policies for OpenStack users, disactivate users after long inactivity (*Cloud*)

[R3] Use a Central Directory for Authentication and Authorization.

- Use an OS authentication system that is bound to LDAP/kerberos/active directory/freeipa ((*Virtual*)
- Enable LDAP for all domains in Keystone (*Cloud*)
- Deploy VMs authenticated through a centralized authentication system (*User*)

[R4] Configure Firewalls to Restrict Access.

- Enable iptables, minimize allowed IPs and ports to necessary services only, do not manually tamper iptables once configured (*Virtual*)
- Do not deploy any VM associated with security groups that allow public access on specific ports (*User*)

[R5] Use TLS/SSL where Possible.

- All services and communications (MySQL, rabbitmq, LDAP, OpenStack services) should be accessible over encrypted channels only (*Virtual*, *Cloud*)
- Every application offered by VMs should offer services over TLS/SSL channels (*User*)

[R6] Do Not Use Default Self-Signed Certificates. All certificates should be signed by a certification authority (*Virtual*, *Cloud*, *User*)

[R7] Configure Centralized Remote Logging.

- Store all logs from system and OpenStack in two different remote logging systems (*Virtual*, *Cloud*)
- Set up their own remote logging system for their applications (*User*)

[R8] Maintain Time Synchronization Services. All nodes/VMs should have the time syncronization system enabled and should use the same network-time server list (*Virtual*, *Users*)

[R9] Review and Minimize Hypervisor Attack Surface. Identify and run a security benchmark against the used hypervisors, disable memory de-duplication (*Virtual*)

[R10] Review and Minimize Virtual Machine Manager Attack Surface.

- Execute vulnerability scans of the virtual machine monitor (i.e., QEMU/KVM) (*Virtual*)
- Execute vulnerability scans of OpenStack services and APIs (*Cloud*)

[R11] Use Templates to Deploy Virtual Machines. Execute vulnerability scans of public images on Glance and check if signature verification is enabled (*Cloud*)

[R12] Disconnect unauthorized devices from Virtual Machines. Disable all unauthorized/unused device ports such as NIC, USB or serial ports, disable compute unified device architecture (CUDA) and direct memory access (DMA) (*Virtual*)

[R13] Disable MAC Address Changes and Promiscuous Mode on Guests. Hypervisor or Network virtualizators should deny MAC address changes on the Vnic (*Virtual*)

[R14] Ensure Network Isolation through VLANs.

- Only VLAN or VLANX should be available and enabled in the whole deployment (*Virtual*)
- Only VLAN or VLANX should be enabled in Neutron (*Cloud*)

[R15] Port Groups Should not be Configured to Reserved VLANs. Neutron m2l plugin should be set to allow only VLAN ids that do not overlap the reserved ones used by physical devices in the network infrastructure (*Cloud*)

[R16] Secure Access to Cloud Application Programming Interfaces. Enable and configure SELinux for secure access to configuration file, run vulnerability scans for OpenStack APIs, isolate API endpoints, especially those with public access, deploy API endpoints on separate hosts for increased isolation (if possible), enable multi-factor authentication (if available) and only provide APIs over SSL/TLS with mutual authentication (Virtual, Cloud)

[R17] Encrypt Data at Rest.

- Nodes should have encrypted file systems (*Virtual*)

- Enable LUKS block storage creation in Cinder and use an appropriate fixed_key as passphrase. Enable encryption feature in Swift configuration file, use an encryption_root_secret that is a base-64 encoding of a 32 byte value generated by a cryptographically secure random number generator (*Cloud*)

[R18] Establish Appropriate Resource Isolation.

- Disable memory de-duplication, avoid co-resident attack, do not allow overlapping of VLAN ids and of virtual disks assignment to hosts, disable live migration or limit migration to dedicated network with encryption enabled, disable delay delete feature for Glance, disable the compute soft delete for Nova, allow to publish public images by admin users only (*Virtual*, *Cloud*)
- Ensure that only allowed users are members of your project, use encrypted storage for all sensitive data (*User*)

[R19] Evaluate Denial of Service Scenarios and Mitigations.

- Mitigation systems should be place in front of critical assets. Rate-limiting from application server should be configured, IDS should be installed and configured to detect DDoS attacks, blacklisting systems for SSH connection should be enabled (e.g., fail2ban) (*Virtual*)
- Run performance test and do not go under resource quotas (*Cloud*)

[R20] Do Not Use or Set Guest Customization Passwords.

- Every node should allow access through a centralized system only; extra users should not exist, unless the necessary ones (*Virtual*)
- Admin should not be member of any project and a policy should not allow the admin to access project resources she is not member of. Admin should not be allowed to set/change user password (*Cloud*)

[R21] Evaluate and Minimize Cloud Architecture Dependencies.

- Hypervisors should be always up and available, hardware resources such RAM and CPUs should be always available and all services such as rabbitmq, MySQL should be running. In addition, high availability should be set for services. (*Virtual*)
- Guarantee high availability of all OpenStack services. In Glance, do not allow creation of unsupported image type. Provide multi-region deployment (*Cloud*)
- Use scheduler filtering to deploy VMs that provide high availability on at least two different availability zones. (*User*)

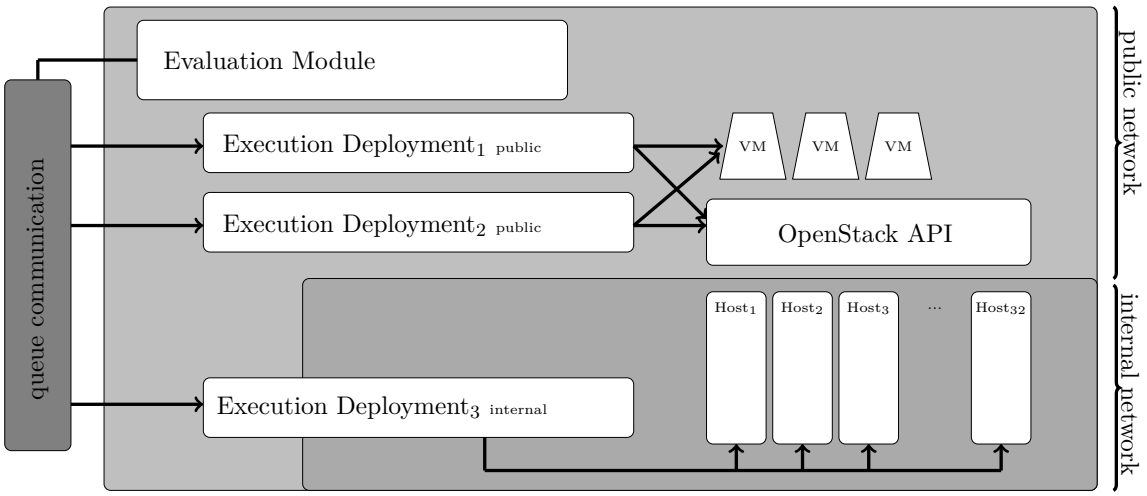


Fig. 6.2 Moon Cloud architecture

[R22] Audit Sensible and Configuration Files. All regular Linux file system and system calls, and OpenStack service configurations should be audited (e.g., auditctl) (*Virtual*, *Cloud*)

[R23] Storage Reliability.

- All OSs should run at least on Raid type 1 to guarantee data replication (*Virtual*)
- Cinder and Swift should use a dedicated storage distributed at least over three replicas (*Cloud*)

[R24] Data Remanence Avoidance. All resources, such as virtual network, block devices, images, should be always bound to entity in corresponding databases to avoid data remanence [136] (*Virtual*)

6.1.3 Security Controls

This section shows how our benchmark can be evaluated against a large, in-production installation of OpenStack using Moon Cloud platform. The target of evaluation is the OpenStack Mitaka deployment at University of Milan, called Lagrange, currently used for research projects and teaching activities. Section 5.4 described in details Moon Cloud, Figure 6.2 shows a summary representation of Moon Cloud architecture and how it was deployed to run the benchmark. One Evaluation module and one Execution module are installed to access OpenStack APIs and VMs (public deployment); one Execution module is installed within the internal network where all Lagrange nodes are reachable (internal deployment).

The goal of the evaluation in this section is to show the applicability and utility of our benchmark on a real OpenStack deployment using a representative subset of recommendations extracted from the benchmark in Section 6.1.2. In the following, for each recommendation,

we provide *i)* ToE t including the target profile, *ii)* control flow ϕ and the corresponding parameters λ associated with the flow's operations, *iii)* environmental settings π, *iv)* a discussion on the evaluation results and a remediation when needed. We note that, all the scripts implementing controls, parameters and environmental settings are available in Appendix D.

[R8] Maintain Time Synchronization Services.
Profile: Virtual.
ToE: All nodes that compose the OpenStack deployment.
Control: The control needs to access every node, and check if the time synchronisation is enabled and every node is connected to the same server list as required. The control supports both crony and ntp.

The execution flow ϕ consists of three sequential operations with the corresponding parameters λ as follows.

1. *connect_to_server* [username, password, private_key, private_key_passphrase,hostname, port]: it accesses to the node through SSH;
2. *check_timesync_enabled* [ntp,chrony]: it checks, using the init system, if crony or ntp is enabled;
3. *check_timesync_config* [ntp_config_file (optional),chrony_config_file (optional),servers_list]: it checks whether servers list in the crony or ntp config file are the same as the ones in the parameters.

. The environmental settings π are as follows:

- Control must be executed with access to the internal network.
- Paramiko python library for SSH connection.

Results: During our evaluation we found that Lagrange is not compliant with this recommendation; in particular, the control returns a negative result for Host 5 because it is using a different pool of time servers from what expected. As remediation, the Host 5 time server configuration file can be replaced with a file that adheres to the recommendations.

[R20] Do Not Use or Set Guest Customization Passwords.
Profile: Cloud.
ToE: Openstack Keystone. Keystone is the identity service and manages projects, users, and groups.
Control: The control requires that admin cannot be member of any projects, excepts her owns projects, and cannot change users' passwords. Hence, the control is twofold: *i)* admin user is only member of a restricted list of projects as specified in a list, *ii)* the OpenStack

policy does not allow admin to change users' password, which must be changed only through the centralized identity system.

The execution flow ϕ of control *i)* consists of two sequential operations with the corresponding parameters λ as follows.

1. *openstack_connection* [os_username, os_password, os_project_id, os_auth_url, os_user_domain_name] it connects to OpenStack APIs using the admin credentials
2. *checkProject* [project_list]: it parses all projects and checks that admin is member only of the projects given as input.

The environmental settings π are as follows:

- Control must be executed with access to the public OpenStack APIs.
- The OpenStack client SDK must be able to communicate with its APIs.

The execution flow ϕ of control *ii)* consists of two sequential operations with the corresponding parameters λ as follows.

1. *connect_to_server* [username, password, private_key, private_key_passphrase, hostname, port]: it accesses the Keystone nodes through SSH.
2. *retrieve_policy_file* [path]: it reads and parses the policy file.
3. *inspect_policy_file* [key, expected_value]: it checks that identity:change_password action is disabled.

The environmental settings π are as follows:

- Control must be executed with access to the internal network.
- Paramiko python library for SSH connection.

Results: Lagrange is fully compliant with this recommendation; the *change_password* is disabled and the admin user is only member of projects *admin* and $admin - test$.

[R21] Evaluate Cloud Architecture Dependencies.
Profile: User.
ToE: Nova computing and user VMs
Control: User can mitigate the risks introduced by a single point of failure by deploying VMs in different availability zones; hence, the control checks that a set of VMs are at least deployed in two different availability zones. The execution flow ϕ of the first control C_1 consists of three sequential operations with the corresponding parameters λ as follows.

1. *openstack-connection* [user credentials]: it accesses OpenStack APIs using user credentials.

2. *retrieve-zone* []: it retrieves all availability zones in OpenStack.

3. *check-deployment* [vm-list]: it checks that at least one VM from vm-list is deployed in a different availability zone.

The environmental settings π are as follows:

- Control must be executed with access to the public OpenStack APIs.
- The OpenStack client SDK to be able to communicate with its APIs.

Results: Lagrange is not compliant with this recommendation since it offers only one availability zone. As remediation, the admin can identify, if possible, fault-independent zones and aggregate hosts under these zones. If not possible the admin can re-design or extend the node cluster to provide different availability zones.

6.2 How to evaluate a Web Hosting Service: AgID compliance

The diffusion of ICT devices and systems in most of the working environments brings many advantages in terms of offered services, high standards of living, automation and performance. The price we pay for ICT benefits is the increasing number of security incidents and breaches, which nowadays worries all the companies, and in turn their customers, with an increasing risk of failures of critical services, data breaches and privacy violations.

In 2016, the cost of data breaches was 3.62 billion dollars, with an increase in their dimension. This value refers only to what has been disclosed by the companies, which are normally reluctant to reveal such security breaches for reputation reasons. The real monetary values is much higher if we also consider the reputation damage suffered by the attacked companies. Some well-known recent incidents resulted in dangerous denial of service, such as the block of hospital services across England and Scotland [137]; other incidents affected the privacy of terabytes of customer transactions (e.g., [138]). In this scenario, according to marketsandmarkets.com [139], enterprise governance, risk management and compliance (GRC) market is becoming one of the most prominent market worldwide and is expected to reach 44 billion dollars in revenue by 2020. It includes all solutions aimed to make companies aligned with the increasing number of mandatory regulations and standards (e.g., European GDPR) in all business sectors. The need of GRC is getting more and more pressing especially in the IT domain. In particular, according to Gartner [140], IT-GRC market is expected to reach 7.3 billion dollars revenue by 2020.

IT security is one of the key aspects driving GRC and one of the major sources of risk perceived by enterprises. According to SANS Institute, the percentage of IT budget that enterprises devote to security is increasing in every domain and is between 4% and 12%; 56%

of this budget is then used to guarantee compliance to regulations, which is among the major factors of IT security expenses. In this context, a report from Kaspersky Lab [141] shows that the average financial loss due to IT incidents is about 31k euro/year for each small/medium enterprise (SME) and about 450k euro/year for each average enterprise. According to CIS Sapienza [142], the budget used by a small enterprise in Italy to protect against IT threats is 41k euro every 5 years (8.3k euro/year), including both setup expenses and recurrent yearly expenses. This budget is 76% lower with respect to the average loss over 5 years. For bigger enterprises, the budget used to protect against IT threats raises to 103k euro every 5 years (20.7k euro per year), 41% lower with respect to the average loss over 5 years. Then, in Europe, with the new privacy regulation (GDPR) becoming effective in 2018, an additional increase of 16% is forecast for the IT security and compliance market.

The market is slowly noticing that it is not the technological inadequacy of security systems that increases the risk of data theft or breach; rather, misconfigurations and faulty integration of such systems within the business processes are at the basis of such thefts/breaches (72% of the total of observed problems).

> *Install the best security controls does not guarantee a secure system; it is necessary to implement a continuous diagnostic process that verifies whether controls are configured in a proper way and behave as expected (security compliance).*

Security assessment [143] becomes then critical especially in cloud and IoT environments. This assessment must be continuous and holistic, to correlate the evidence collected by an increasing number of security protection mechanisms. Koschorreck [144] described the importance of automated audit and security controls with particular attention to models and standard protocols to represent and exchange audit results and operations. Alzahrani et al. [145] presented an interesting analysis of the available web application security tools from SQL injection to black box vulnerability scanners. Patel et al. [146] presented an analysis of vulnerabilities and security of the most common CMS such as WordPress and Joomla.

In this Section, we show how Moon Cloud can be used to continuously evaluate the status of a web hosting service. The evaluation is carried out according to the ICT security guidelines for Italian public administration provided by the *Agenzia per l'Italia Digitale* (AgID) and approved by the Italian prime minister.

6.2.1 The scenario

The evaluation covers a web hosting service provide by University of Milan and it is carried out according to the ICT security guidelines for Italian public administration provided by the *Agenzia per l'Italia Digitale* (AgID) and approved by the Italian prime minister.

The web hosting service, which is provided to all the employees of an enterprise and span different (from application to infrastructure) layers of the ICT system. We define a complete compliance evaluation process, which verifies a wide variety of services, from official company web sites and single project/professional workspaces, to hosting web servers and virtualized infrastructure. In particular, the multi-layer system target of our evaluation provides access to the web sites hosted on two web servers. The web servers are installed in virtual machines with a Linux operating system (e.g., CentOS) deployed on a virtualized infrastructure based on VMware. Administration privileges are granted to employees by means of an access control system based on *cPanel*, while the updated list of employees is maintained in an active directory.

Manage such a complex scenario is not a trivial operation, since the responsibility of content, technology and updates belongs to different owners at different levels, that is, web site owners, service (e.g., web server) administrators, and infrastructure administrators. This sharing of responsibility, and in turn of liability, cause loss of control by the enterprise, which is not aware of the status of its infrastructure and the impact that buggy web sites could have on the whole infrastructure. In addition, the separation between the management of employees and administration privileges substantially increases the risk of unauthorized requests, harnessing the system security. A critical goal for the enterprise is therefore to establish a level of control on the status of the web sites through an activity of classification and vulnerability assessment, which also involves lower levels of the systems, in order to provide a complete view on the status of the security to all involved parties. Such goal can be accomplished manually, though it requires huge amount of time and impairs repeatability due to unbearable management costs. Moon Cloud platform permits to achieve a complete, automatic and repeatable assessment at four layers: [A1] web site vulnerability, [A2] operating system security, [A3] authorization process correctness, [A4] virtualization security. Each layer corresponds to a security compliance policy.

6.2.2 AgID

Agenzia per l'Italia Digitale (AgID) is the agency of the Italian government responsible for defining and continuously updating the IT security requirements for the Italian public administration. It defines the AgID Basic Security Controls (ABSC), starting from the controls defined in SANS 20 that are now published by the Center for Internet Security "CIS Critical Security Controls for Effective Cyber Defence". AgID Basic Security Controls are organized in 13 categories for a total of 121 controls, and are categorized in three levels of strength, minimum, standard, high.

We selected an excerpt of these controls that are relevant for the web hosting service described in Section 6.2.1 and belong to four main categories: ABSC2 - authorized and non-authorized software inventory; ABSC3 - protect hardware and software configurations

Table 6.2 AgID Basic Security Controls (Excerpt)

Category	ABSC_ID	Description	Mapping to Assessment Layers
ABSC2	2.1.1	Write a list of authorized software and corresponding versions for each system, including servers, workstations and laptops. Forbid installation of software not in the list	A1, A2
ABSC2	2.3.1	Execute regular scan on the system to identify non-authorized software	A1, A2, A4
ABSC3	3.1.1	Use standard and secure operating system configurations	A2
ABSC3	3.1.2	Standard and secure configurations must correspond to the hardened version of the operating system and installed applications	A2
ABSC3	3.2.1	Identify and use standard and secure configurations for workstations, servers and other systems	A2
ABSC4	4.1.1	Execute a vulnerability assessment at each configuration change	A1, A2, A4
ABSC4	4.1.2	Execute a vulnerability assessment recurrently depending on the criticality of the asset	A1, A2, A4
ABSC4	4.1.3	Use a Security Content Automation Protocol (SCAP) to find both code-based vulnerabilities and configuration-based vulnerabilities	A1, A2, A4
ABSC4	4.5.1	Automatically install software patches and updates both for operating systems and applications,	A2, A4
ABSC5	5.2.1	Maintain the inventory of all administrators, guaranteeing that each administrator is authorized	A3

of mobile devices, laptops, workstations, and servers; ABCS4 - continuous vulnerability assessment; ABCS5 - correct use of administrator privileges. Table 6.2 summarizes the ABSC target of our evaluation presenting their category, the identifier of the specific AgID Basic Security Control, and its description. Each control is also mapped to the corresponding assessment layers in Section 6.2.1: [A1] web site vulnerability, [A2] operating system security, [A3] authorization process correctness, [A4] virtualization security.

6.2.3 Security Controls

We present the Moon Cloud security controls (C) for the identified four assessment layers. For each control, we provide information about its goal representing the purpose of the control, its target representing the component under evaluation, its implementation referring to the sequential operations and input needed to execute the control, and its mapping to the ABSC in Table 6.2. For each assessment layer, in Table 6.3, we also present how the corresponding security controls are integrated in a security compliance policy for compliance evaluation. Probes' code described in this section are not reported in Appendix D since belong to Moon Cloud intellectual property.

Table 6.3 AgID Basic Security Controls (Excerpt)

Assessment layer	Security Compliance Policy	Description
A1	P1=C1∧C2∧(C3∨C4)	The policy verifies the risk associated with Joomla and WordPress web site using a vulnerability scanner.
A2	P2=C5∧C6∧C7	The policy verifies the robustness of the operating systems installed on the two virtual machines offering the web servers.
A3	P3=C8∧C9∧C10	The policy verifies the consistency between access privileges and existing authorized users.
A4	P4=C11	The policy verifies the security of the virtualization layer based on VMware.

Web site vulnerability assessment [A1]

It is composed of three steps: *i)* check web site availability, *ii)* retrieve web site information, and *iii)* execute a vulnerability scan. These three steps are automated and sequentially executed to *i)* identify web sites which are up and running, *ii)* among web sites at point *i)*, identify which ones use a CMS (Content Management System), Wordrpress, or Joomla,[2] and *iii)* run a dedicated scan against web sites classified at point *ii)*.

Web Site Availability Checker [C1]. When a node hosts hundreds of web sites, which are managed by different owners, a web site availability checker is needed to verify whether a web site is up and running. The evidence that some of them are not running can be an indicator of a possible security threat that needs to be investigated, moreover this tool helps in the asset inventory maintenance. The unavailability of a web site can be verified by evaluating the HTTP response code.

- *Goal.* The goal of the control is to check whether the web site is available or not.
- *Target.* Web sites.
- *Implementation.* The evaluation is carried out using the HTTP protocol. The control executes a HTTP GET operation on a target URL (usually the index page). Based on the server response, the web site is marked as available or not. HTTP error codes can inform that a website does not exist, code 404, or that there is an internal server error, code 5xx, or that there is a misconfiguration in the DNS and the url cannot be resolved, code 105. All these codes entail a failed test result.
- *AgID Controls.* ABSC2 2.1.1 , ABSC2 2.3.1.

Web Site Info Collector [C2]. A web site info collector is fundamental to identify the technologies at the basis of every web site. This control identifies frameworks, languages and CSM, to determine whether forbidden software/applications are used and keep track of all installed versions. It will then help in the choice of the most appropriate vulnerability

[2]The choice of integrating and describing Joomla and Worpress vulnerability scanners is based on the analysis of the web sites available in the experimental environment used in the next section.

scan. This control is based on the open source tool wappalyzer https://wappalyzer.com/ and can provide information organized in 53 categories that cover cache tool, Database, CSM, Javascript Framework, and many others.

- *Goal.* The goal of the control is to provide extensive information on the web technologies used by each web site and check that no forbidden installed software/applications are used.
- *Target.* Web sites.
- *Implementation.* The control first sanitizes the website URL passed as input and then runs wappalyzer against the web site. All results are organized based on the wappalyzer categories and listed for discovery and inventory purposes. The probe returns *true* if all installed software/applications are in the list of authorized software/applications, while it returns *false* if at least one software/application does not belong to the list of authorized software/applications or is not classified yet.
- *Agid Controls.* ABSC2 2.1.1 , ABSC2 2.3.1.

WordPress Vulnerability Scanner [C3]. WordPress (https://wordpress.org/) is one of the most popular CMS in the world and is used by nearly 75 million websites, powering the 25% of the world's website. Given its large usage, Moon Cloud integrates the wp-scan project https://wpscan.org/ that is a black box WordPress vulnerability scanner.

- *Goal.* The goal of the control is to check whether the WordPress web sites have vulnerabilities.
- *Target.* Web sites.
- *Implementation.* The evaluation is carried out using wp-scan. The control first checks if the web site uses WordPress as CSM and then runs the wp-scan vulnerability scanner against it. Once the scan is over, if vulnerabilities are found, they are collected and classified by CVSS (Common Vulnerability Scoring System) and the evaluation returns false; otherwise, if no vulnerabilities are found, the evaluation returns true. Since web sites often suffer from dated vulnerabilities from 2010 or even from 2007, we used CVSS version 2.0.
- *AgID Controls.* ABSC4 4.1.1, ABSC4 4.1.2, ABSC4 4.5.1.

Joomla Vulnerability Scanner [C4]. Joomla https://www.joomla.org/ is another largely used CMS. Moon Cloud integrates the joomla-scanner https://github.com/rezasp/joomscan, also supported by the OWASP community. Joomla-scanner is a dedicated black box Joomla vulnerability scanner, which provides a specific assessment for Joomla templates, components, versions, and modules.

- *Goal.* The goal of the control is to check whether the Joomla web sites have vulnerabilities.
- *Target.* Web sites.
- *Implementation.* The evaluation is carried out using joomla-scanner. The control, first of all, checks if the web site uses Joomla as CSM and then runs the joomla-scanner vulnerability scanner against it. Once the scan is over if vulnerabilities are found, they are collected and classified by CVSS (Common Vulnerability Scoring System) and the evaluation returns *false*; otherwise if no vulnerabilities are found, the evaluation returns *true*. Again, we used CVSS version 2.0.
- *Agid Controls.* ABSC4 4.1.1, ABSC4 4.1.2, ABSC4 4.5.1.

Policy P1 in Table 6.3 presents how the controls for assessment layer A1 are integrated in a security compliance policy.

Operating System Security Assessment [A2]

It is composed of three steps that are executed on every VM: *i)* running a certified SCAP using a standard checklist , *ii)* list vulnerable packages that need to be updated, and *iii)* search for unauthorized packages. These three steps permit to checks the robustness of the operating system installed on the two virtual machines offering the web servers. In other words, they permit to monitor the operating systems and keep them updated and hardened to avoid possible attacks due to service misconfigurations.

OSCAP FedRAMP [C5]. Configuring a host and keep updated is a trivial task, more-over when a team manage different host for different purpose. Even automated installing approaches (e.g. puppet, ansible) may lead to some misconfigurations. this probe runs OpenScap (https://www.open-scap.org/), awarded the SCAP 1.2 certification by NIST, to validate Centos 7 Operating System based on the NIST security guidelines (NIST SP 800-53).

- *Goal.* The goal of the control is to check whether the operating system is compliant to standard NIST SP 800-53.
- *Target.* Operating System CentOS 7.
- *Implementation.* The evaluation is carried out using OpenScap. After installing OpenScap on the target host, the probe connects to it using SSH and injects the SCAP Security Guide (SGG) including the security policies to be verified. SGG includes a large set of test cases (357) that may result in *i)* pass, *ii)* fail, *iii)* not checked or *iv)* not applicable. The result is parsed and the probe returns a success evaluation if there are no failed test cases.

- *AgID Controls.* ABSC3 3.1.1, ABSC3 3.1.2, ABSC3 3.2.1, ABSC4 4.1.3.

Vulnerable Packages [C6]. Every Linux distro usually has its own package manager, which helps in managing (install, update and uninstall) software packages. This probe analyzes using the distro package manager all the installed software and checks for security updates.

- *Goal.* The goal of the control is to check whether the operating system does not have any security updates to be installed.
- *Target.* Operating System CentOS 7.
- *Implementation.* The evaluation is carried out using the yum package manager, which has a convenient option returning the list of security updates available and not yet applied to the target host. The list is parsed to extract all the CVEs. If there is at least one security update not installed the probe returns *false*, otherwise *true*.
- *AgID Controls.* ABSC4 4.1.1, ABSC4 4.1.2.

Authorized Packages [C7]. With a set of hosts to manage, having the possibility to continuously check for installed software allows to keep the asset inventory updated and to check whether unauthorized software is installed.

- *Goal.* The goal of the control is to check whether every host has installed authorized software only.
- *Target.* Operating System CentOS 7.
- *Implementation.* The evaluation is carried out using the yum package manager, which provides a list of all installed software. This list is mapped to the asset inventory to check for compliance. If there is a package that is not listed or is unauthorized the probe returns *false*, otherwise *true*.
- *AgID Controls.* ABSC2 2.1.1, ABSC2 2.3.1, ABSC4 4.5.1

Policy P2 in Table 6.3 presents how the controls for assessment layer A2 are integrated in a security compliance policy.

Authorization Process Security Assessment

It is composed of three steps: *i)* check virtualization layer access list, *ii)* check host layer access list, and *iii)* check web site management access list. These three steps are automated and executed to keep, where possible, consistent user management at different system layers.

VMware ESXI user list [C8]. VMware ESXI is a multi-tenant OS at the basis of the web hosting system. It can be bound to the LDAP to allow only authorized users to have access to the admin console.

- *Goal.* The goal of the control is to check whether admin users are listed as employee in the LDAP.
- *Target.* VMware ESXI.
- *Implementation.* The evaluation is carried out using the VMware API; the probe retrieves the list of all users and checks their role. This list is then compared to the employee database to check for inconsistencies.
- *AgID Controls.* ABSC5 5.2.1.

Host user list [C9]. Accesses to a given host should be granted to authorized users only. Direct access to the host should be granted for management purpose only and authorized by a support ticket. This control checks whether all accesses in a given time frame were authorized or not.

- *Goal.* The goal of the control is to check that every access was granted by a support ticket.
- *Target.* VM host.
- *Implementation.* The evaluation is carried out accessing the access logs. Using the command *utmpdump* on the logs /var/logs/wtemp is possible to analyze all accesses collecting information about access time, user and IP source. The list of accesses is then matched with all tickets related to the target VM. If there is at least a mismatch the probe returns *false*; otherwise, *true*.
- *AgID Controls.* ABSC5 5.2.1.

Web Site Privileges [C10]. Every employee can have one or more web sites, and be responsible for their management and maintenance. The web admin manages a database mapping employees to available web sites. The databases is not directly connected to the human resource IT system, therefore it may lead to data inconsistency. To avoid scenarios where former employees retain privileges on web sites installed on the host, a continuous consistency check between the HR and web admin databases is executed.

- *Goal.* The goal of the control is to check that only active employees have access to the web site service.
- *Target.* Web Site service.
- *Implementation.* The evaluation is carried out accessing the web admin databases. For each user in the web admin database there must be a corresponding user in the HR database. If all user listed in the web admin database are also found in the HR database, the probe returns *true*; otherwise, it returns *false*.

- *AgID Controls.* ABSC5 5.2.1.

Policy P3 in Table 6.3 presents how the controls for assessment layer A3 are integrated in a security compliance policy.

Virtualization Security Assessment

It checks the robustness of the underlying virtualization infrastructure, to guarantee the security of the whole system. The virtualization layer is based on VMware ESXI. The control follows the CIS benchmark for VMware as a guideline for running all evaluations.

VMware CIS benchmark [C11]. The CIS benchmark for VMware covers different categories that goes from logging to network and disk management. It also prescribes to analyze the VMs configurations.

- *Goal.* The goal of the control is to evaluate VMWare ESXi based on the CIS Benchamark.
- *Target.* VMware ESXi.
- *Implementation.* The evaluation is carried out using the VMware API. The probe executes the audit operations needed to verify each recommendation in the CIS document.[3] If all recommendations are satisfied the probe returns *true*; otherwise, *false*.
- *AgID Controls.* ABSC3 3.1.1, ABSC3 3.1.2, ABSC4 4.1.3

Policy P4 in Table 6.3 consists of control C11 only.

6.2.4 Security Controls

We run the controls against the web hosting service of the Università degli Studi di Milano (UNIMI), one of the biggest universities in Europe. We note that, for confidentiality reasons, only the control for web site vulnerability assessment has been executed on the in-production web hosting service of UNIMI. All other controls have been executed on a simulated environment different from the in-production UNIMI web hosting service. Also, the retrieved experimental results have been filtered out due to their criticality and sensitivity.

Moon Cloud platform was installed on a Virtual Machine with 2 cores, 4GB of RAM, 10GB of ephemeral disk and 60GB of persistent disk. To reduce the additional load observed by UNIMI servers, Moon Cloud platform was configured to schedule and execute controls over a period of 24 hours. This configuration has been specified through the Moon Cloud dashboard and made the process completely automatic. The results of the evaluation process are reported below.

[3] We note that the probe runs only the subset of CIS recommendations that can be automated.

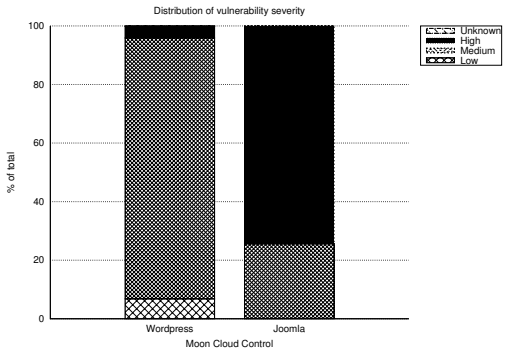


Fig. 6.3 Frequency distribution (%) of found CVEs over WordPress and Joomla web sites classified by CVSS families

Web site vulnerability assessment

The assessment process was composed of the following phases: *i)* run control *Web Site Availability Checker* to list the web sites that are available and those that are unreachable; *ii)* run control *Web Site Info Collector* to make an inventory of the web technologies used by each web site and identify those web sites that use either Worpress or Joomla; *iii)* run control *Wordpress Vulnerability Scan* on all WordPress web sites; *iv)* run control *Joomla Vulnerability Scan* on all Joomla web sites.

Moon Cloud analyzed web siteshosted by UNIMI; 82.5% of them was up and running, while 17.5% was unreachable. Among them, 25.3% of the web sites use Wordress, while 5.4% web sites use Joomla. The remaining 69.6% of web sites consists of custom PHP or static HTML web pages.

We remark that, in our analysis, we used CVSS 2.0 scoring system. In particular, CVSS 2.0 assigns scores from 0 to 10 to each Common Vulnerability Exposure (CVE) and can be organized in 4 main families: *Low*, *Medium*, *High*, and *Critical*. CVSS score is retrieved from the NIST National Vulnerability Database (https://nvd.nist.gov/). Figure 6.3 shows the distribution of the found vulnerabilities (CVE) based on their CVSS score for WordPress and Joomla web sites.

For web sites using WordPress, found vulnerabilities were distributed as follows: 3.9% of vulnerability has been classified as high, 89.1% as medium, 6.8% as low, over 92 different CVEs. 0.2% of vulnerabilities have been classified as not scored because they refer to two reserved CVEs. Figure 6.4(a) shows the frequency distribution (percentage) of found CVEs over Wordpress web sites. It is important to note that all analyzed web sites were vulnerable to CVE-20**-****, because no countermeasure was provided at evaluation time (October 2017).

The same analysis was carried out on Joomla web sites. Figure 6.3 shows the distribution of found vulnerabilities based on their CVSS. Among found vulnerabilities, 25.5% has been

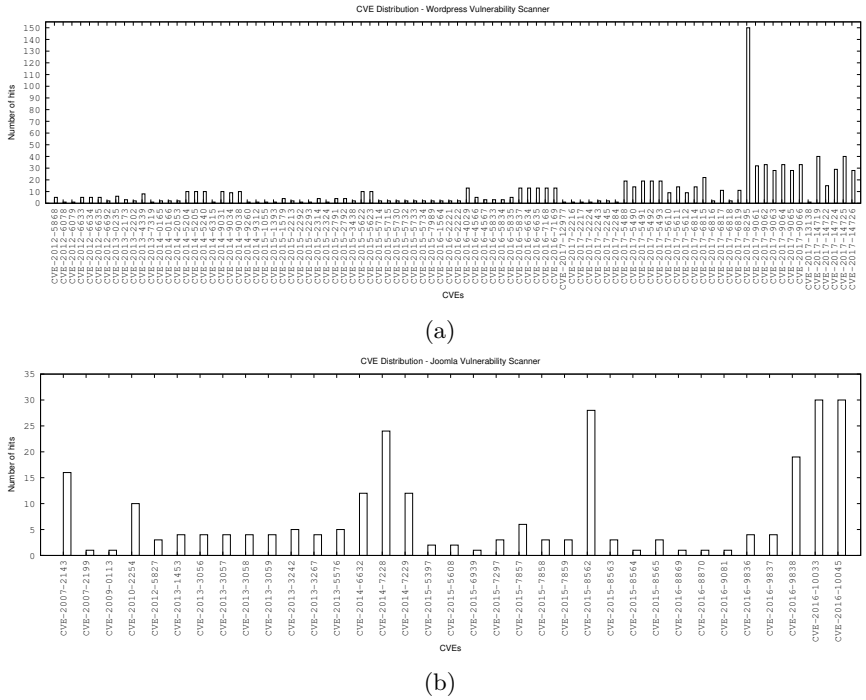


Fig. 6.4 Frequency distribution (%) of found CVEs over Wordpress (a) and Joomla (b) web sites. We note that, for the sake of readability, Figure 6.4(a) only reports CVEs that affected at least three WordPress web sites. CVE details have been anonymized for security reasons

classified as medium and 74.5% as high, over 35 different CVEs. Two common vulnerabilities, as presented in Figure 6.4(b), affected the 93.7% of analyzed web sites.

Policy P1 in Table 6.3 is evaluated for each web site. We note that all WordPress web sites retrieved a negative evaluation of P1 due to zero-day vulnerability CVE-20**-****; 94.7% of Joomla web sites retrieved a negative evaluation of P1 due to two main vulnerabilities. It is important to note that all found vulnerabilities are not at the same level of criticality. The result of the evaluation of policy P1 could also be weighted on the basis of the found vulnerability; for instance, low risk vulnerabilities might not cause P1 evaluation to fail.

To conclude, we note that, in order to run our experiments, we deactivated a perimeter firewall having the role of blocking malicious traffic and reducing the impact of zero-day vulnerabilities. We also note that the web site owners sign a term of service with UNIMI becoming responsible for web site management, updates, patches, and the like. This makes even more evident how this sharing of responsibilities makes security management even more critical and difficult.

Other assessment layers

We briefly summarize the results retrieved by executing the controls in the remaining three assessment layers on a simulated environment as follows.

- *Operating System Security Assessment.* We run controls C5, C6, C7 over two virtual machine installing CentOS operating system. C6 and C7 returned a positive result since all packages in the simulated environment were updated and authorized; C8, which executed 190 single checks corresponding to NIST 800-171 requirements, returned a negative result since 67 out of 190 checks failed. The severity of the fails were 10 low, 47 medium, and 10 high. For the sake of simplicity, we only present three high severity checks found by Moon Cloud:
  - Ensure gpgcheck Enabled for Local Packages: yum was not configured to verify the signature(s) of local packages prior to installation.
  - Prevent Log In to Accounts With Empty Password: an account had an empty password; anyone could then log in and run commands with the privileges of that account.
  - Disable SSH Access via Empty Passwords: this setting was not configured for the SSH daemon, making remote login via SSH with empty password possible.

  Policy P2 then returned a negative evaluation.
- *Authorization Process Security Assessment.* We run controls C8, C9, C10 showing that there was full consistency between the list of employees in the user database and the admin accounts at virtualization, host, and web site layers. Policy P3 then returned a positive evaluation.
- *VMware Security Assessment.* The CIS benchmark for VMware (C11) was executed on the simulated virtualization layer retrieving a positive evaluation. All checks were correct and policy P4 then returned a positive evaluation.

6.3 Chapter Summary

We presented a certification framework that implements a security certification process for the cloud as described in Chapter 3 and 4. Our framework supports pre-deployment and production evaluation of cloud systems at different layers of the cloud stack, it provides a certification-aware cloud engineering methodology that drives providers in the design and development of ready-to-be-certified systems. Our approach departs from traditional certification schemes which are independent from the development process, to provide a certification scheme with reduced overheads. We designed different probes to meet the different evaluation and collection activities and integrate in the framework a Big Data Platform to collect and process huge amount of data.

Moreover, we presented Moon Cloud a spin-off of Universitá degli Studi di Milano fully compatible with the framework and built on the outcomes of this work.

Chapter 7

Conclusion

In this thesis we have investigated assurance and certification techniques for cloud services. After a brief introduction, and related work (Chapters 1-2), Chapter 3 described our idea of cloud certification process and the cloud certification scheme designed to satisfy the process. Based on Chapter 3, Chapter 4 extended the certification scheme to fit certification of cloud composite services; moreover, it proposed a solution to minimize costs of cloud composite certification deployment. Chapter 5 described the framework designed and developed to meet the requirements identified during the analysis of cloud service certification. Chapter 6 presented two different scenarios where the framework was successfully applied. In this last chapter, we shortly summarize the contribution of this thesis and outline some topics left to future work.

7.1 Summary of the contributions

The contributions of this thesis is manifold.

Cloud Service Certification. We analyzed how cloud service certification should be addressed, identifying issues, requirements and solutions. We proposed a cloud certification process that includes an automatic and incremental approach to certificate adaptation, addressing the multi-layer and dynamics nature of the cloud. We designed a Certification Model Template (CMT) and a Certification Model Instance (CMI) that fully describe property, target and activities of a certification. We defined a consistency check algorithm that permits to automatically verify if a CMI is a right instantiation of a CMT. The consistency check algorithm is fundamental to build a safe chain of trust grounded on service certification. We developed two heuristics that accomplish the consistency check providing good performance with a reasonable accuracy.

Certification of Cloud Composite Service Cloud permits to compose single services to form a more complex cloud composite service that achieves the user's functional requirements. Certification of single cloud services does not fit the scenario where the final service is a composite service that may change at any time. We proposed a method for cloud composite service certification that aims to minimize any re-certification operation permitting to obtain a new certificate from the artifacts (certificates and corresponding evidence) of the component services. Our certification scheme supports migration and versioning.

Cost-effective Deployment of Cloud Composite Service We have extended and adapted the work on cloud service certification to meet the requirements of cloud composite services. We analyse the deployment of cloud composite servicea from the point of view of the cloud provider taking into account *direct, mismatch and certification* costs. In this context we defined three cost-profiles (i.e. sharing, average and fitting) and a cost-effective algorithm that aims to minimize the deployment cost of cloud composite service based on non-functional requirements. We developed a first heuristic based on a sliding window of size k that analyzes the best deployment for the composite services within the window. We defined a second heuristic, as an extension of the first, which supports service migration. We analyzed both heuristics in terms of performance and quality varying k and cost-profile.

Cloud Certification Framework We proposed a certification framework that implements a security certification process for the cloud. Our framework supports pre-deployment and in-production evaluation of cloud systems at different layers of the cloud stack. It also provides a certification-aware cloud engineering methodology that drives providers in the design and development of ready-to-be-certified systems. We presented Moon Cloud platform, a running solution for security governance and compliance assessment. Moon Cloud platform supports continuous verification, diagnostic, and monitoring of ICT system compliance against security policies. It provides an enhanced methodology that permits to correlate different and heterogeneous evidence to evaluate the status of security in a given system. We showed how assurance techniques, provided by Moon Cloud, can evaluate two different cloud scenarios: *i)* a full security evaluation of the IaaS manager OpenStack based on a specific security benchmark, *ii)* a security evaluation of a web hosting service, which analyzes the full cloud stack from the vmware esxi hypervisor hosting the web server nodes to the application level corresponding to the available web sites.

7.2 Future works

The research described in this thesis can be extended along different directions, to accomplish the requirements and peculiarities of new evolving distributed paradigms and environments.

We identify three main scenatios that have been increasing considered in the last few years: IoT and Fog, Big Data, and Machine Learning.

IoT and Fog Fog Computing and IoT (internet of things) share with cloud computing their dynamic and distributed nature. An IoT formal definition is proposed by the ISO/IEC: *"infrastructure of interconnected objects, people, systems and information resources together with intelligent services to allow them to process information of the physical and the virtual world and react" [147]* . However cloud common characteristics, IoT presents a more challenging scenario [148, 149]:

- low computational capacity may restrict the type of applicable assurance techniques;
- constraints on battery consumption may limit type and frequency of evaluation activities;
- placement and network configuration may make devices unreachable or required some ad-ho probe deployment;
- fuzzy perimeters and high variability requires a certification life-cycle management even lighter and more responsive;
- heterogenous devices both in technology and purpose lead current solution to certification and their generality.
- distributed liability makes the identification of the responsible parties for unattended behaviour difficult to identify.

Moreover, fog adds an extra layer between cloud and IoT, this highlight how the three technologies are strictly related and certification should take into account the whole IoT supply chain.

Big Data A Big data platform is already part of the framework and new analytics addressing new and challenging scenario are valuable contributions. However big data analytics is used as an enabler for security evaluation, big data security and privacy are increasingly becoming the target of different research and development efforts in terms of big data assurance [150, 151]. A number of different solutions have been provided to protect the Big Data infrastructures and their data/processes by internal and external threats and attacks. These efforts resulted in the proliferation of ad hoc security and privacy solutions, which prove a specific security property or compliance to regulations. In this scenario, our certification models could be used to assess that big data infrastructure and analytics are acting under specific non-functional requirements extending the 5V definition of Big Data towards 6V where the 6th is *verification*:

- *Volume* refers to the vast amounts of data generated every second that increasingly makes data sets too large to be stored and analyzed using traditional database approaches.

- *Velocity* refers to the speed at which new data are generated and moved.
- *Variety* refers to the different types of data (structured, unstructured) that can be used and correlated together.
- *Value* the data collected should be turned into tangible value.
- *Veracity* refers to the messiness or trustworthiness of the data. Today, the data processing part cannot be discarded and introduces the need of verifying Big Data trustworthiness as well, introducing the following 6th V.
- *Verification* refers to the possibility of verifying the assurance of a Big Data process, increasing its transparency and trustworthiness. For instance, it verifies that a given Big Data engine satisfies the performance requirements or treats data with an adequate privacy level (e.g., GDPR compliance).

A 6V Big Data process carries a higher level of trust that makes it suitable for critical Big Data scenarios, as for instance the ones implementing a generic assurance process.

Machine Learning In this thesis we assumed that both the Certification Authority and Accredit Labs can identify the right evaluation activities for every target and every property. This task requires an enormous effort due to the heterogeneity of cloud services and the high number of properties available. Assurance activities might be reused in different certification models, but it still strictly depends on the skills of the evaluator. Machine learning might enhance this process by suggesting the proper assurance techniques for a given property and a given target. In this direction it is important to be able to describe precisely all the three concepts: *i)* a generic evaluation activity, *ii)* the property and *iii)* the target. Even if evidence collection is fully described as an STS including input, expected output and input required, both property and target do not currently support such detailed description.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," in *Tech. Rep. UCB/EECS-2009-28*, EECS Department, U.C. Berkeley, February 2009.

[2] S. Lins, S. Schneider, and A. Sunyaev, "Trust is good, control is better: Creating secure clouds by continuous auditing," *IEEE TCC*, vol. PP, no. 99, pp. 1–1, 2016.

[3] I. Windhorst and A. Sunyaev, "Dynamic certification of cloud services," in *Proc. of ARES 2013*, Regensburg, Germany, September 2013.

[4] M. Anisetti, C. Ardagna, E. Damiani, and F. Saonara, "A test-based security certification scheme for web services," *ACM TWEB*, vol. 7, no. 2, pp. 1–41, May 2013.

[5] Microsoft, *Trusted Cloud*, https://www.microsoft.com/en-us/server-cloud/trusted-cloud/overview.aspx, Accessed June 2016.

[6] M. Anisetti, C. Ardagna, E. Damiani, and F. Saonara, "A test-based security certification scheme for web services," *ACM Transactions on the Web*, vol. 7, no. 2, pp. 1–41, May 2013.

[7] B. Bertholon, S. Varrette, and P. Bouvry, "Certicloud: A novel tpm-based approach to ensure cloud IaaS security," in *Proc. of IEEE CLOUD 2011*, Washington, DC, USA, July 2011.

[8] S. Pearson, "Toward accountability in the cloud," *IEEE Internet Computing*, vol. 15, no. 4, pp. 64–69, 2011.

[9] H. Rasheed, "Data and infrastructure security auditing in cloud computing environments," *International Journal of Information Management*, December 2013.

[10] E. Damiani, C. Ardagna, and N. E. Ioini, *Open source systems security certification.* New York, NY, USA: Springer, 2009.

[11] A. Munoz and A. Mãna, "Bridging the gap between software certification and trusted computing for securing cloud computing," in *Proc. of IEEE SERVICES 2013*, June 2013, pp. Santa Clara, CA, USA.

[12] G. Spanoudakis, E. Damiani, and A. Maña, "Certifying services in cloud: The case for a hybrid, incremental and multi-layer approach," in *Proc. of IEEE HASE 2012*, Omaha, NE, USA, October 2012.

[13] S. Lins and A. Sunyaev, "Unblackboxing IT Certifications: A Theoretical Model Explaining IT Certification Effectiveness," in *Proc. of ICIS 2017*, Seoul, South Korea, Dec. 2017, pp. 1–13, keyword: Certification.

[14] R. Medeiros, N. S. Rosa, and L. F. Pires, "Predicting service composition costs with complex cost behavior," in *Proc. of IEEE SCC*, New York, NY, USA, June 2015.

[15] Q. He, J. Han, F. Chen, Y. Wang, R. Vasa, Y. Yang, and H. Jin, "Qos-aware service selection for customisable multi-tenant service-based systems: Maturity and approaches," in *Proc. of IEEE CLOUD*, New York, NY, USA, June-July 2015.

[16] P. Leitner, W. Hummer, and S. Dustdar, "Cost-based optimization of service compositions," *IEEE TSC*, vol. 6, no. 2, pp. 239–251, April 2013.

[17] X. Bai, M. Li, B. Chen, W.-T. Tsai, and J. Gao, "Cloud testing tools," in *Proc. of IEEE SOSE 2011*, Irvine, CA, USA, December 2011.

[18] J. Gao, X. Bai, W.-T. Tsai, and T. Uehara, "Saas testing on clouds - issues, challenges and needs," in *Proc. of IEEE SOSE 2013*, San Francisco, CA, USA, March 2013.

[19] G. Aceto, A. Botta, W. De Donato, and A. Pescapè, "Cloud monitoring: A survey," *Comput. Netw.*, vol. 57, no. 9, pp. 2093–2115, Jun. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2013.04.001

[20] S. Meng and L. Liu, "Enhanced monitoring-as-a-service for effective cloud management," *IEEE TC*, vol. 62, no. 9, pp. 1705–1720, September 2013.

[21] M. Anisetti, C. Ardagna, and E. Damiani, "A certification-based trust model for autonomic cloud computing systems," in *Proc. of ICCAC 2014*, London, UK, September 2014.

[22] Z. Xiao and Y. Xiao, "Security and privacy in cloud computing," *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 843–859, Second 2013.

[23] C. B. O. M. E. Moctar and K. Konate, "A survey of security challenges in cloud computing," in *Proc. of WiSPNET 2017*, March 2017, pp. 843–849.

[24] B. Qin, H. Wang, Q. Wu, J. Liu, and J. Domingo-Ferrer, "Simultaneous authentication and secrecy in identity-based data upload to cloud," *Cluster Computing*, vol. 16, no. 4, pp. 845–859, Dec 2013. [Online]. Available: https://doi.org/10.1007/s10586-013-0258-7

[25] S. Ruj, M. Stojmenovic, and A. Nayak, "Decentralized access control with anonymous authentication of data stored in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 384–394, Feb 2014.

[26] L. Wei and M. K. Reiter, "Ensuring file authenticity in private dfa evaluation on encrypted files in the cloud," in *in Proc. of ESORICS 2013*, J. Crampton, S. Jajodia, and K. Mayes, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 147–163.

[27] P. Sha and Z. Zhu, "The modification of rsa algorithm to adapt fully homomorphic encryption algorithm in cloud computing," in *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, Aug 2016, pp. 388–392.

[28] H. Teigeler, S. Lins, and A. Sunyaev, "Chicken and egg problem: What drives cloud service providers and certification authorities to adopt continuous service certification?" in *Proceedings of the Pre-ICIS Workshop on Information Security and Privacy*, Association for Information Systems (AIS). AIS, Dezember 2017, Inproceedings.

[29] E. Council, *Information Assurance*, 201u, http://www.consilium.europa.eu/en/general-secretariat/corporate-policies/classified-information/information-assurance/.

[30] *ISO/IEC 21827:2008 Information technology – Security techniques – Systems Security Engineering – Capability Maturity Model SSE-CRM*, 2008.

[31] *ISO/IEC24765 Systems and software engineering – Vocabulary*, 2017, https://www.ecs.csus.edu/csc/iac/cnssi_4009.pdf.

[32] IATAC and DACS, *Software Security Assurance: State of the Art Report (SOAR)*, 2007, http://www.dtic.mil/dtic/tr/fulltext/u2/a472363.pdf.

[33] C. on National Security Systems (CNSS), *National Information Assurance (IA) Glossary instruction No. 4009*, 2006, https://www.ecs.csus.edu/csc/iac/cnssi_4009.pdf.

[34] S. Quirolgico, J. Voas, T. Karygiannis, C. Michael, and K. Scarfone, *NIST Special Publication 800-163 Vetting the Security of Mobile Applications*, 2015.

[35] *SOFTWARE ASSURANCE STANDARD - NASA TECHNICAL STANDARD*, 2004, https://standards.nasa.gov/standard/nasa/nasa-std-87398.

[36] J. G. Cooper and K. A. Pauley, "Healthcare software assurance," in *Proc. of AMIA 2006*, 2006. [Online]. Available: http://knowledge.amia.org/amia-55142-a2006a-1.620145/t-001-1.623243/f-001-1.623244/a-033-1.623649/a-034-1.623646

[37] W. A. Conklin, "Software assurance: The need for definitions," *in Proc of HiCSS 2011*, pp. 1–7, 2011.

[38] C. Ardagna, R. Asal, E. Damiani, and Q. Vu, "From security to assurance in the cloud: A survey," *ACM CSUR*, vol. 48, no. 1, pp. 2:1–2:50, August 2015.

[39] E. V. Veenendaal, *Standard Glossary of Terms used in Software Testing*, International Software Testing Qualifications Board (ISTQB), 2012, http://www.consilium.europa.eu/en/general-secretariat/corporate-policies/classified-information/information-assurance/.

[40] C. Wu and S. Marotta, "Framework for assessing cloud trustworthiness," in *Proc. of Cloud 2013*, June 2013, pp. 956–957.

[41] P. Zech, "Risk-based security testing in cloud computing environments," in *Proc. of ICST 2011*, Berlin, Germany, March 2011.

[42] G. S. D. Oliveira and A. Duarte, "A framework for automated software testing on the cloud," in *Proc. of PDCAT 2013*, Dec 2013, pp. 344–349.

[43] L. Gao, M. Lu, L. Li, and C. Pan, "A survey of software runtime monitoring," in *in Proc of ICSESS 2017*, Nov 2017, pp. 308–313.

[44] K. Mahbub and G. Spanoudakis, *Monitoring WS-Agreements: An Event Calculus–Based Approach.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 265–306. [Online]. Available: https://doi.org/10.1007/978-3-540-72912-9_10

[45] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience." *Parallel Computing*, vol. 30, no. 5-6, pp. 817–840, 2004. [Online]. Available: http://dblp.uni-trier.de/db/journals/pc/pc30.html#MassieCC04

[46] S. A. D. Chaves, R. B. Uriarte, and C. B. Westphall, "Toward an architecture for monitoring private clouds," *IEEE Communications Magazine*, vol. 49, no. 12, pp. 130–137, December 2011.

[47] J. M. A. Calero and J. G. Aguado, "Monpaas: An adaptive monitoring platformas a service for cloud computing infrastructures and services," *IEEE Transactions on Services Computing*, vol. 8, no. 1, pp. 65–78, Jan 2015.

[48] C. B. Hauser and S. Wesner, "Reviewing cloud monitoring: Towards cloud resource profiling," in *Proc. of CLOUD 2018*, July 2018, pp. 678–685.

[49] K. Alhamazani, R. Ranjan, K. Mitra, F. Rabhi, P. P. Jayaraman, S. U. Khan, A. Guabtni, and V. Bhatnagar, "An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art," *Computing*, vol. 97, no. 4, pp. 357–377, Apr 2015. [Online]. Available: https://doi.org/10.1007/s00607-014-0398-5

[50] S. Pearson, "Toward accountability in the cloud," *IEEE Internet Computing*, vol. 15, no. 4, pp. 64–69, Jul. 2011. [Online]. Available: http://dx.doi.org/10.1109/MIC.2011.98

[51] A. Haeberlen, "A case for the accountable cloud," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 52–57, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773926

[52] J. Agarkhed and R. Ashalatha, "An efficient auditing scheme for data storage security in cloud," in *2017 International Conference on Circuit ,Power and Computing Technologies (ICCPCT)*, April 2017, pp. 1–5.

[53] S. Hiremath and S. Kunte, "A novel data auditing approach to achieve data privacy and data integrity in cloud computing," in *Proc. of ICEECCOT 2017*, Dec 2017, pp. 306–310.

[54] D. K. Konoor, "Auditing in cloud computing solutions with openstack," in *2016 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, Oct 2016, pp. 176–176.

[55] T. Indhumathil, N. Aarthy, V. D. Devi, and V. N. Samyuktha, "Third-party auditing for cloud service providers in multicloud environment," in *Proc. of ICONSTEM 2017*, March 2017, pp. 347–352.

[56] H. Rasheed, "Data and infrastructure security auditing in cloud computing environments," *Int J. Information Management*, vol. 34, no. 3, pp. 364–368, 2014.

[57] *ISO/IEC Guide 2:2004 Standardization and related activities – General vocabulary*, 1996.

[58] *IISO/IEC JTC 1/SC 27 IT Security techniques*, 1989.

[59] *DoD, Department Of Defense Trusted Computer System Evaluation Criteria. USA Department of Defence*, December 1985, http://csrc.nist.gov/publications/secpubs/rainbow/std001.txt.

[60] M. Krotsiani, G. Spanoudakis, and K. Mahbub, "Incremental certification of cloud services," in *Proc. of SECURWARE 2013*, Barcelona, Spain, August 2013.

[61] *CSA Security, Trust & Assurance Registry (STAR)*, Cloud Security Alliance (CSA), https://cloudsecurityalliance.org/star/, Accessed June 2016.

[62] M. Anisetti, C. Ardagna, and E. Damiani, "A low-cost security certification scheme for evolving services," in *Proc. of ICWS 2012*, Honolulu, HI, USA, June 2012.

[63] C. Criteria, *CCRA Supporting Document 2004-02-009 Assurance Continuity*, February 2004, http://www.commoncriteriaportal.org/files/supplements/2004-02-009.pdf.

[64] J. Zhang, "A mobile agent-based tool supporting web services testing," *Wireless Personal Communications*, vol. 56, no. 1, pp. 147–172, Jan 2011. [Online]. Available: https://doi.org/10.1007/s11277-009-9879-9

[65] D. Herrmann, *Using the Common Criteria for IT security evaluation.* Auerbach Publications, 2002.

[66] D. Kourtesis, E. Ramollari, D. Dranidis, and I. Paraskakis, "Increased reliability in SOA environments through registry-based conformance testing of web services," *Production Planning & Control*, vol. 21, no. 2, pp. 130–144, 2010.

[67] A. Sunyaev and S. Schneider, "Cloud services certification," *Communications of the ACM*, vol. 56, no. 2, pp. 33–36, February 2013.

[68] X. Chen, C. Chen, Y. Tao, and J. Hu, "A cloud security assessment system based on classifying and grading," *IEEE Cloud Computing*, vol. 2, no. 2, pp. 58–67, 2015.

[69] P. Stephanow, G. Srivastava, and J. Schütte, "Test-based cloud service certification of opportunistic providers," in *Proc. of CLOUD 2016*, San Francisco, CA, USA, July–June 2016.

[70] P. Stephanow and N. Fallenbeck, "Towards continuous certification of Infrastructure-as-a-Service using low-level metrics," in *Proc. of ATC 2015*, Beijing, China, August 2015.

[71] M. Krotsiani, G. Spanoudakis, and C. Kloukinas, "Monitoring-based certification of cloud service security," in *Proc. of C&TC 2015*, Rhodes, Greece, October 2015.

[72] M.Dekker, C. Karsberg, and D. L. M. Lakka, *Auditing Security Measures,An Overview of schemes for auditing security measures*, ENISA, Septemberr 2013, http://csrc.nist.gov/publications/secpubs/rainbow/std001.txt.

[73] R. K. L. Ko, P. Jagadpramana, M. Mowbray, S. Pearson, M. Kirchberg, Q. Liang, and B. S. Lee, "Trustcloud: A framework for accountability and trust in cloud computing," in *Proc. of IEEE SERVICES 2011*, Washington, DC, USA, 2011, pp. 584–588.

[74] Q. Malluhi and K. M. Khan, "Establishing trust in cloud computing," *IT Professional*, vol. 12, pp. 20–27, 2010.

[75] A. Naskos, A. Gounaris, H. Mouratidis, and P. Katsaros, "Online analysis of security risks in elastic cloud applications using probabilistic model checking," *(To appear in) IEEE Cloud Computing Magazine*, 2016.

[76] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "Towards trustworthy multi-cloud services communities: A trust-based hedonic coalitional game," *IEEE TSC*, vol. PP, no. 99, pp. 1–1, 2016.

[77] F. Fittkau, S. Frey, and W. Hasselbring, "Cdosim: Simulating cloud deployment options for software migration support," in *Proc. of MESOCA 2012*, Sept 2012, pp. 37–46.

[78] S. Brumec and N. VrčEk, "Cost effecti veness of commercial computing clouds," *Inf. Syst.*, vol. 38, no. 4, pp. 495–508, Jun. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.is.2012.11.002

[79] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: Comparing public cloud providers," in *Proc. of IMC 2010*, Melbourne, Australia, November 2010.

[80] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008.

[81] X. Li, Y. Li, T. Liu, J. Qiu, and F. Wang, "The method and tool of cost analysis for cloud computing," in *in Proc of CLOUD 2009*, Sept 2009, pp. 93–100.

[82] D. Worm, M. Zivkovic, H. van den Berg, and R. van der Mei, "Revenue maximization with quality assurance for composite web services," in *Proc. of IEEE SOCA 2012*, Taipei, Taiwan, December 2012.

[83] C. Di Giulio, C. Kamhoua, R. H. Campbell, R. Sprabery, K. Kwiat, and M. N. Bashir, "It security and privacy standards in comparison: Improving fedramp authorization for cloud service providers," in *Proc. of the CCGrid 2017*, ser. CCGrid '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 1090–1099. [Online]. Available: https://doi.org/10.1109/CCGRID.2017.137

[84] C. D. Giulio, R. Sprabery, C. Kamhoua, K. Kwiat, R. H. Campbell, and M. N. Bashir, "Cloud standards in comparison: Are new security frameworks improving cloud security?" in *Proc. of Cloud 2017*, June 2017, pp. 50–57.

[85] J. Becker and E. Bailey, "A comparison of it governance and control frameworks in cloud computing," in *Proc. of AMCIS 2014*, 2014.

[86] *FedRAMP. Guide to Understanding FedRAMP, v2. 2014.*, 2014.

[87] *ISO/IEC 27001 - Information security management*, ISO/IEC, November 2015, http://www.iso.org/iso/home/standards/management-standards/iso27001.htm.

[88] T. Saxena and V. Chourey, "A survey paper on cloud security issues and challenges," in *Proc. of CSIBIG 2014*, March 2014, pp. 1–5.

[89] X. Jing and Z. Jian-jun, "A brief survey on the security model of cloud computing," in *Proc. of DCABES 2010*, Aug 2010, pp. 475–478.

[90] S. Rajeswari and R. Kalaiselvi, "Survey of data and storage security in cloud computing," in *Proc. of ICCS 2017*, Dec 2017, pp. 76–81.

[91] A. Aich, A. Sen, and S. R. Dash, "A survey on cloud environment security risk and remedy," in *Proc. of CINE 2015*, Jan 2015, pp. 192–193.

[92] W. C. N. Kaura and A. Lal, "Survey paper on cloud computing security," in *Proc. of ICIIECS 2017*, March 2017, pp. 1–6.

[93] A. Waqas, Z. M. Yusof, and A. Shah, "A security-based survey and classification of cloud architectures, state of art and future directions," in *Proc. ACSAT 2013*, Dec 2013, pp. 284–289.

[94] J. Zhang, L. Zheng, L. Gong, and Z. Gu, "A survey on security of cloud environment: Threats, solutions, and innovation," in *Proc. of DSC 2018*, June 2018, pp. 910–916.

[95] *Top Threats to Cloud Computing Plus: Industry Insights*, Cloud Security Alliance (CSA), 2018, https://cloudsecurityalliance.org/download/top-threats-cloud-computing-plus-industry-insights/.

[96] R. Barona and E. A. M. Anita, "A survey on data breach challenges in cloud computing security: Issues and threats," in *Proc. ICCPCT 2017*, April 2017, pp. 1–8.

[97] D. Kolevski and K. Michael, "Cloud computing data breaches a socio-technical review of literature," in *Proc. of ICGCIoT 2015)*, Oct 2015, pp. 1486–1495.

[98] S. Kirkman, "A data movement policy framework for improving trust in the cloud using smart contracts and blockchains," in *Proc. of IC2E 2018*, April 2018, pp. 270–273.

[99] S. Jajodia, W. Litwin, and T. Schwarz SJ, *Recoverable Encryption through a Noised Secret over a Large Cloud*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 42–64. [Online]. Available: https://doi.org/10.1007/978-3-642-40069-8_3

[100] R. Fathi, M. A. Salehi, and E. L. Leiss, "User-friendly and secure architecture (ufsa) for authentication of cloud services," in *Proc. of CLOUD 2015*, June 2015, pp. 516–523.

[101] A. J. Choudhury, P. Kumar, M. Sain, H. Lim, and H. Jae-Lee, "A strong user authentication framework for cloud computing," in *Proc. of APSCC 2011*, Dec 2011, pp. 110–115.

[102] R. B. Bahaweres and J. S. M. Alaydrus, "Building a private cloud computing and the analysis against dos (denial of service) attacks: Case study at smkn 6 jakarta," in *Proc. of CITSM 2016*, April 2016, pp. 1–6.

[103] R. Gracia-Tinedo, M. S. Artigas, and P. G. Lopez, "Cloud-as-a-gift: Effectively exploiting personal cloud free accounts via rest apis," in *2013 IEEE Sixth International Conference on Cloud Computing*, June 2013, pp. 621–628.

[104] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip feng shui: Hammering a needle in the software stack," in *Proc. of USENIX 2016*. Austin, TX: USENIX Association, 2016, pp. 1–18. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi

[105] M. M. Hasan and M. A. Rahman, "Protection by detection: A signaling game approach to mitigate co-resident attacks in cloud," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, June 2017, pp. 552–559.

[106] X. Liang, X. Gui, A. N. Jian, and D. Ren, "Mitigating cloud co-resident attacks via grouping-based virtual machine placement strategy," in *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, Dec 2017, pp. 1–8.

[107] D. Capelli, A. Moore, R. Trzeciak, and T. J. Shimeall, *Common Sense Guide to Prevention and Detection of Insider Threats, 3rd ed*, CERT), 2009, http://www.cert.org/archive/pdf/CSG-V3.pdf.

[108] L. Nkosi, P. Tarwireyi, and M. O. Adigun, "Detecting a malicious insider in the cloud environment using sequential rule mining," in *Proc. of ICASTech 2013*, Nov 2013, pp. 1–10.

[109] F. Rocha, T. Gross, and A. v. Moorsel, "Defense-in-depth against malicious insiders in the cloud," in *2013 IEEE International Conference on Cloud Engineering (IC2E)*, March 2013, pp. 88–97.

[110] A. N. Rukavitsyn, K. A. Borisenko, I. I. Holod, and A. V. Shorov, "The method of ensuring confidentiality and integrity data in cloud computing," in *Proc. of SCM 2017*, May 2017, pp. 272–274.

[111] K. N. Sevis and E. Seker, "Survey on data integrity in cloud," in *Proc. of CSCloud 2016*, June 2016, pp. 167–171.

[112] C. Consortium, *D2.1: Security-aware SLA specification language and cloud security dependency model*, http://cumulus-project.eu/index.php/public-deliverables, Accessed in Date March 2016.

[113] *Health Insurance Portability and Accountability Act (HIPAA)*, U.S. Department of Health & Human Services, November 2015, http://www.hhs.gov/ocr/privacy/hipaa/understanding/.

[114] M. Anisetti, C. Ardagna, and E. Damiani, "A test-based incremental security certification scheme for cloud-based systems," in *Proc. of SCC 2015*, New York, NY, USA, June–July 2015.

[115] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE TC*, vol. 100, no. 8, pp. 677–691, 1986.

[116] S.-Y. Hwang, E.-P. Lim, C.-H. Lee, and C.-H. Chen, "Dynamic web service selection for reliable web service composition," *IEEE TSC*, vol. 1, no. 2, pp. 104–116, April 2008.

[117] M. Anisetti, C. Ardagna, and E. Damiani, "Security certification of composite services: A test-based approach," in *Proc. of the 20th IEEE International Conference on Web Services (ICWS 2013)*, San Francisco, CA, USA, June-July 2013.

[118] F. Tao, D. Zhao, Y. Hu, and Z. Zhou, "Resource service composition and its optimal-selection based on particle swarm optimization in manufacturing grid system," *IEEE TII*, vol. 4, no. 4, pp. 315–327, November 2008.

[119] L. Qi, W. Dou, X. Zhang, and J. Chen, "A qos-aware composition method supporting cross-platform service invocation in cloud environment," *J. Comput. Syst. Sci.*, vol. 78, no. 5, pp. 1316–1329, September 2012.

[120] H. Kurdi, A. Al-Anazi, C. Campbell, and A. Al Faries, "A combinatorial optimization algorithm for multiple cloud service composition," *Comput. Electr. Eng.*, vol. 42, no. C, pp. 107–113, February 2015.

[121] X. Wang, J. Zhu, and Y. Shen, "Network-aware qos prediction for service composition using geolocation," *IEEE TSC*, vol. 8, no. 4, pp. 630–643, July 2015.

[122] K. Kofler, I. u. Haq, and E. Schikuta, "User-centric, heuristic optimization of service composition in clouds," in *Proc. of Euro-Par 2010*, Ischia, Italy, August-September 2010.

[123] Jula, Amin, Sundararajan, Elankovan, Othman, and Zalinda, "Cloud computing service composition: A systematic literature review," *Expert Systems with Applications*, vol. 41, no. 8, pp. 3809–3824, 2014.

[124] C. T. Horngren, G. Foster, S. M. Datar, M. Rajan, C. Ittner, and A. A. Baldwin, "Cost accounting: A managerial emphasis," *Issues in Accounting Education*, vol. 25, no. 4, pp. 789–790, 2010.

[125] R. W. de Medeiros, N. S. Rosa, and L. F. Pires, "Predicting service composition costs with complex cost behavior," in *Services Computing (SCC), 2015 IEEE International Conference on.* IEEE, 2015, pp. 419–426.

[126] S. Newman, *Building Microservices*. O'Reilly Media, Inc., 2015.

[127] C. Sadtler, Z. X. Chen, S. Imazeki, M. Kelm, S. Kofkin-Hansen, Z. Q. Kou, B. McChesney *et al.*, *IBM Workload Deployer: Pattern-based Application and Middleware Deployments in a Private Cloud*. IBM Redbooks, 2012.

[128] I. Maleki, L. Ebrahimi, S. Jodati, and I. Ramesh, "Analysis of software cost estimation using fuzzy logic," *International Journal in Foundations of Computer Science & Technology (IJFCST)*, vol. 4, no. 3, pp. 27–41, 2014.

[129] C. A. Ardagna, V. Bellandi, P. Ceravolo, E. Damiani, M. Bezzi, and C. Hebert, "A model-driven methodology for big data analytics-as-a-service," in *Proc. of 2017 BigData Congress*, June 2017, pp. 105–112.

[130] Z. Hu, L. Zhu, C. Ardi, E. Katz-Bassett, H. V. Madhyastha, J. Heidemann, and M. Yu, "The need for end-to-end evaluation of cloud availability," in *Passive and Active Measurement*, M. Faloutsos and A. Kuzmanovic, Eds. Cham: Springer International Publishing, 2014, pp. 119–130.

[131] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using docker technology," in *Proc. of SoutheastCon 2016*, March 2016, pp. 1–5.

[132] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," in *Tech. Rep. UCB/EECS-2009-28*, EECS Department, U.C. Berkeley, February 2009.

[133] "Cis security benchmark repository," https://benchmarks.cisecurity.org/downloads/.

[134] OpenStack Foundation, *OpenStack Security Guide*, April 2015, http://docs.openstack.org/security-guide/security-guide.pdf.

[135] G. Gerchow, M. A. Haines, and P. Goyal, "Cis quick start cloud infrastructure benchmark v1.0.0," October 2012, https://benchmarks.cisecurity.org/downloads/show-single/?file=cloud.100/.

[136] B. Albelooshi, K. Salah, T. Martin, and E. Damiani, "Experimental proof: Data remanence in cloud vms," in *Proc. IEEE CLOUD2015*, June 2015.

[137] BBC, *NHS cyber-attack: GPs and hospitals hit by ransomware*, May 2017, http://www.bbc.com/news/health-39899646.

[138] N. Perlroth, *All 3 Billion Yahoo Accounts Were Affected by 2013 Attack*, October 2017, https://www.nytimes.com/2017/10/03/technology/yahoo-hack-3-billion-users.html.

[139] marketsandmarkets.com, *eGRC Market by Component (Services, Software), Deployment Mode (Cloud, On-Premises), Business Function (Finance, It, Legal, Operations), Organization Size, Vertical, Usage, and Region - Global Forecast to 2022*, July 2017, https://media.kaspersky.com/pdf/it-risks-survey-report-cost-of-security-breaches.pdf.

[140] John A. Wheeler, *IRM Solutions Market Will Grow to $7.3 Billion by 2020*, March 2017, https://blogs.gartner.com/john-wheeler/irm-solutions-market-will-grow-to-7-3-billion-by-2020/.

[141] Karpersky Lab, *Damage Control: The Cost of Security Breaches*, 2015, https://media.kaspersky.com/pdf/it-risks-survey-report-cost-of-security-breaches.pdf.

[142] CIS Sapienza, *2016 Italian Cybersecurity Report*, March 2017, http://www.cybersecurityframework.it/sites/default/files/csr2016web.pdf.

[143] J. Modic, R. Trapero, A. Taha, J. Luna, M. Stopar, and N. Suri, "Novel efficient techniques for real-time cloud security assessment," *COSE*, vol. 62, pp. 1–18, 2016.

[144] G. Koschorreck, "Automated audit of compliance and security controls," in *2011 Sixth International Conference on IT Security Incident Management and IT Forensics*, May 2011, pp. 137–148.

[145] A. Alzahrani, A. Alqazzaz, Y. Zhu, H. Fu, and N. Almashfi, "Web application security tools analysis," in *Proc. of IEEE bigdatasecurity 2017*, May 2017.

[146] S. K. Patel, V. R. Rathod, and J. B. Prajapati, "Comparative analysis of web security in open source content management system," in *Proc. of ISSP 2013*, March 2013, pp. 344–349.

[147] *Information Technology. Internet of things (iot). preliminary report.*, ISO/IEC, 2014.

[148] C. A. Ardagna, E. Damiani, J. Schütte, and P. Stephanow, *A Case for IoT Security Assurance*. Singapore: Springer Singapore, 2018, pp. 175–192. [Online]. Available: https://doi.org/10.1007/978-981-10-5861-5_8

[149] P. Beaudou, P. lBradley, E. Clement, R. Cricco, C. Dietze, E. Laffont, D. Lopez, R. M. Gonzalez, D. Vujcic, and T. Wozniak, *An analysis of the security needs of the 5G market*, SimAlliance, 2018.

[150] R. Cao and J. Gao, "Research on reliability evaluation of big data system," in *2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, April 2018, pp. 261–265.

[151] P. Zhang, X. Zhou, W. Li, and J. Gao, "A survey on quality assurance techniques for big data applications," in *Proc. of BigDataService 2017*, April 2017, pp. 313–319.

Appendix A

Publication

This work has appeared in varying forms, in the form of journals and conference papers. In the following we report papers and articles published or submitted used in this thesis.

1. **A Certification Framework for Cloud-based Services**
   Co-author: M. Anisetti, C.A. Ardagna, E. Damiani
   in Proc. of SAC 2016, April 2016, Pisa, Italy
   Abstract: Lack of trust and transparency are among the main reasons hindering adoption of cloud computing. Users in fact can inspect neither their applications nor the treatment of their data, and have little or no guarantees about their security. In this context, there is a pressing need for assurance techniques supporting some key properties of cloud services and applications. Cloud security certification is a major assurance technique that has been proposed to increase cloud security, trust, and transparency. However, certification is a tedious, costly, and time-consuming process for the provider that wants to certify one of its services/applications. In this paper, we propose a test-based security certification framework for the cloud implementing a certification process and a cloud engineering methodology based on it, which supports providers in the design and development of ready-to-be-certified services/applications.

2. **A Security Benchmark for OpenStack**
   Co-author: M. Anisetti, C.A. Ardagna, E. Damiani
   in Proc. of IEEE CLOUD 2017, June 2017, Honolulu, HI, USA
   Abstract: The cloud computing paradigm entails a radical change in IT provisioning, which must be understood and correctly applied especially when security requirements are considered. Security requirements do not cover anymore just the application itself, but involve the whole cloud supply chain from the hosting infrastructure to the final applications. This scenario requires, on one side, new security mechanisms protecting the cloud against misbehaviors/malicious attacks and, on the other side, a continuous and adaptive assurance process evaluating the observed cloud security behavior against the expected one. In this paper, we focus on the evaluation of the security assurance of OpenStack, a major open source cloud infrastructure. We first define a security benchmark for OpenStack, inspired by Center for Internet Security (CIS) benchmark for cloud infrastructures. We then present a platform, called Moon Cloud, for cloud

security assurance evaluation, showing an application of our benchmark and platform to the in-production OpenStack deployment of the University of Milan.

3. **A semi-automatic and trustworthy scheme for continuous cloud service certification**

   Co-author: M. Anisetti, C.A. Ardagna, E. Damiani

   in IEEE Transactions on Services Computing, 2016

   Abstract: Traditional assurance solutions for software-based systems rely on static verification techniques and assume continuous availability of trusted third parties. With the advent of cloud computing, these solutions become ineffective since services/applications are flexible, dynamic, and change at run time, at high rates. Although several assurance approaches have been defined, cloud requires a step-change moving current assurance techniques to fully embrace the cloud peculiarities. In this paper, we provide a rigorous and adaptive assurance technique based on certification, towards the definition of a transparent and trusted cloud ecosystem. It aims to increase the confidence of cloud customers that every piece of the cloud (from its infrastructure to hosted applications) behaves as expected and according to their requirements. We first present a test-based certification scheme proving non-functional properties of cloud-based services. The scheme is driven by non-functional requirements defined by the certification authority and by a model of the service under certification. We then define an automatic approach to verification of consistency between requirements and models, which is at the basis of the chain of trust supported by the certification scheme. We also present a continuous certificate life cycle management process including both certificate issuing and its adaptation to address contextual changes. Finally, we describe our certification framework and an experimental evaluation of its performance, quality, applicability, and practical usability in a real industrial scenario, which considers Engineering Ingegneria Informatica S.p.A. ENGpay online payment system.

4. **A Cost-Effective Certification-Based Service Composition for the Cloud**

   Co-author: M. Anisetti, C.A. Ardagna, E. Damiani

   in Proc. of IEEE CLOUD 2016 , June-July 2016, San Francisco, CA, USA

   Abstract: The cloud computing paradigm provides an environment where services can be composed and reused at high rates. Existing composition techniques focus on providing the desired functionality and at a given deployment cost. In this paper, we focus on the definition of cloud service compositions driven by certified non-functional properties. We define a cost evaluation methodology aimed to provide the composition that minimizes the total costs of the cloud provider taking into account deployment, certification, and mismatch costs, and evaluate it using three different cost profiles.

5. **Modeling time, probability, and configuration constraints for continuous cloud service certification**

   Co-author: M. Anisetti, C.A. Ardagna, E. Damiani, N. El Ioini

   Computers & Security (COSE) 72, 2018

   Abstract: Cloud computing proposes a paradigm shift where resources and services are allocated, provisioned, and accessed at runtime and on demand. New business opportunities emerge for service providers and their customers, at a price of an increased uncertainty on how their data are managed and their applications operate once stored/deployed in the cloud.

This scenario calls for assurance solutions that formally assess the working of the cloud and its services/processes. Current assurance techniques increasingly rely on model-based verification, but fall short to provide sound checks on the validity and correctness of their assessment over time. The approach in this paper aims to close this gap catching unexpected behaviors emerging when a verified service is deployed in the target cloud. We focus on certification-based assurance techniques, which provide customers with verifiable and formal evidence on the behavior of cloud services/processes. We present a trustworthy cloud certification scheme based on the continuous verification of model correctness against real and synthetic service execution traces, according to time, probability, and configuration constraints, and attack flows. We test the effectiveness of our approach in a real scenario involving ATOS SA eHealth application deployed on top of open source IaaS OpenStack.

6. **Toward Security and Performance Certification of OpenStack**
   Co-author: M. Anisetti, C.A. Ardagna, E. Damiani, R. Veca
   in Proc. of IEEE CLOUD 2015 , June July 2015, New York City, NY, USA
   textbfAbstract: Cloud users and service providers are increasingly concerned about the management of their data and the behavior of the applications they use/own once stored/deployed in the cloud. They therefore ask for enhanced assurance solutions, which partially mitigate the new risks and threats they are facing. Among existing solutions, certification has been widely adopted as a preferable approach to increase trust in the cloud. In this paper, after briefly discussing our test-based certification scheme for the cloud, we show a real certification process aimed to certify OpenStack, an open source IaaS solution for managing infrastructure resources. In particular, we first describe the testing activities executed to certify OpenStack for security and performance properties. We then illustrate the obtained results and the outcomes of the certification process.

7. **Moon Cloud: A Cloud Platform for ICT Security Governance**
   Co-author: M. Anisetti, C.A. Ardagna, E. Damiani
   in Proc. of IEEE GlobeCom 2018 , December 2018, Abu Dhaby, EAU
   textbfAbstract: Cybersecurity is the second emergency in Europe just after the climate changes. Everyday most of the small, medium and big enterprises are under attack. This scenario requires, on one side, new security solutions protecting ICT systems against misbehaviors/malicious attacks and, on the other side, a continuous assurance process evaluating the system robustness against new threats. In this paper we present Moon Cloud, a Cloud PaaS solution providing customizable assurance based on compliance for ICT systems, including public and private cloud systems and IoT environments. We also present a concrete security assessment carried out in a real scenario.

8. **Cost-Effective Deployment of Certified Cloud Composite Services**
   Co-author: M. Anisetti, C.A. Ardagna, E. Damiani
   under review: Journal of Parallel and Distributed Computing
   textbfAbstract: The cloud computing paradigm provides an environment where services can be composed and reused at high rates. While traditional approaches to service composition are driven by the desired functionality and requirements on deployment costs, more recent approaches also focus on SLAs and non-functional requirements. Service composition in the

cloud introduces new requirements on composition approaches, which need to select component services on the basis of their non-functional properties and continuously adapt to both functional and non functional changes of the component services. These approaches must then depart from the assumption that the cost of the composition is only the sum of the deployment costs of the component services, and also consider the costs of continuously verifying SLAs and non-functional requirements to guarantee a stable Quality of Service (QoS). In this paper, we first present a portable certification process for evaluating non-functional properties of composite services addressing cloud peculiarities. We then focus on the definition of an approach to the composition of cloud services driven by certified non-functional properties. We finally define a cost-evaluation methodology aimed to build the composition that minimizes the total cost paid by the cloud providers, taking into account both deployment and certification/verification costs.

Appendix B

Model Consistency Check

This appendix presents additional information for Chapter 3. It reports an extensive representation, including XML, of examples 3.3.1, 3.3.2. Moreover, SectionB.3 contains the pseudo code of the two consistency model check heuristics. The Dataset of the experimentation due to its length is not reported here, but it as available at https://github.com/SESARLab/tsc-matching.

B.1 Examples

This paragraph contains all artifacts related to Examples 3.3.1, 3.3.2. These two examples represent an example of CM Template and Instance respectively. Other examples that we have used in the context of the CUMULUS project are available at https://github.com/SESARLab/tsc-matching.

B.1.1 Example 3.3.1

Property: Confidentiality

```
<SecurityProperty class="Confidentiality">
  <propertyAttributesList>
    <item>
      in-transit
    </item>
    <item>
      at-rest
    </item>
  </propertyAttributesList>
</SecurityProperty>
```

ToCS: Tocs contains the three mechanisms:

- encryption-message-in-transit
- encryption-internal-communications
- encryption-data-at-rest

```
<ToCs>
  <ToC id="1">
    <mechanism>encryption-message-in-transit</mechanism>
    <service>service</service>
  </ToC>
  <ToC id="2">
    <mechanism>encryption-internal-communications</mechanism>
    <service>service</service>
  </ToC>
  <ToC id="3">
    <mechanism>encryption-data-at-rest</mechanism>
    <service>infrastructure</service>
  </ToC>
</ToCs>
```

STS model

Figure:

Fig. B.1 STS representation

```
<graphml xmlns="http://graphml.graphdrawing.org/xmlns" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns http://
    graphml.graphdrawing.org/xmlns/1.1/graphml.xsd">
  <key id="mechanism" for="node" attr.name="mechanism" attr.type="string" />
  <graph id="G" edgedefault="directed">
```

```
    <node id="n0">
      <data key="mechanism">encryption-message-in-transit</data>
    </node>
    <node id="n1">
      <data key="mechanism">encryption-internal-communications</data>
    </node>
    <node id="n2">
      <data key="mechanism">encryption-data-at-rest</data>
    </node>

    <edge id="0" source="n0" target="n1" label="23497da5-6eba-4528-815d-c70e291f35c7" />
    <edge id="1" source="n1" target="n2" label="dfff5733-3e20-4af1-951d-5a1a585b6c7f" />
  </graph>
</graphml>
```

Evidence

```
<allEvidences>
  <evidences path="0">
    <evidence>
      <mechanism>encryption-message-in-transit</mechanism>
      <state>n0</state>
      <inputs>
        <input>
          <key>host</key>
          <value></value>
          <domain>d1</domain>
        </input>
      </inputs>
      <expectedOutput>
        <output>
          <key>Result</key>
          <value>true</value>
          <domain>d1</domain>
        </output>
      </expectedOutput>
    </evidence>
    <evidence>
      <mechanism>encryption-internal-communications</mechanism>
      <state>n1</state>
      <inputs>
        <input>
          <key>serviceUri</key>
          <value></value>
          <domain>d1</domain>
        </input>
      </inputs>
```

```
        <expectedOutput>
           <output>
              <key>Result</key>
              <value>true</value>
              <domain>d1</domain>
           </output>
        </expectedOutput>
     </evidence>
     <evidence>
        <mechanism>encryption-data-at-rest</mechanism>
        <state>n2</state>
        <inputs>
           <input>
              <key>local</key>
              <value></value>
              <domain>d1</domain>
           </input>
        </inputs>
        <expectedOutput>
           <output>
              <key>Result</key>
              <value>true</value>
              <domain>d1</domain>
           </output>
        </expectedOutput>
     </evidence>
  </evidences>
</allEvidences>
```

B.1.2 Example 3.3.2

Property: Confidentiality

```
<SecurityProperty class="Confidentiality">
  <propertyAttributesList>
    <item>
      in-transit
    </item>
    <item>
      at-rest
    </item>
  </propertyAttributesList>
</SecurityProperty>
```

ToCS: Tocs contains the three mechanisms:

- encryption-XML-encryption-WS-Security-message-in-transit
- encryption-internal-communications-HTTPS
- encryption-at-rest-encrypted-FS

```
<ToCs>
  <ToC id="1">
    <mechanism>encryption-XML-encryption-WS-Security-message-in-transit</mechanism>
    <service>service</service>
  </ToC>
  <ToC id="2">
    <mechanism>encryption-internal-communications-HTTPS</mechanism>
    <service>service</service>
  </ToC>
  <ToC id="2">
    <mechanism>encryption-at-rest-encrypted-FS</mechanism>
    <service>infrastructure</service>
  </ToC>
</ToCs>
```

STS model

Figure:

Fig. B.2 STS representation

```
<graphml xmlns="http://graphml.graphdrawing.org/xmlns" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns http://
    graphml.graphdrawing.org/xmlns/1.1/graphml.xsd">
  <key id="mechanism" for="node" attr.name="mechanism" attr.type="string" />
  <graph id="G" edgedefault="directed">
```

```
    <node id="n0">
      <data key="mechanism">encryption-message-in-transit</data>
    </node>
    <node id="n1">
      <data key="mechanism">encryption-internal-communications</data>
    </node>
    <node id="n2">
      <data key="mechanism">encryption-data-at-rest</data>
    </node>

    <edge id="0" source="n0" target="n1" label="23497da5-6eba-4528-815d-c70e291f35c7" />
    <edge id="1" source="n1" target="n2" label="dfff5733-3e20-4af1-951d-5a1a585b6c7f" />
  </graph>
</graphml>
```

Evidence

```
<?xml version="1.0" encoding="UTF-8"?>
<allEvidences>
  <evidences path="0">
    <evidence>
      <mechanism>encryption-XML-encryption-WS-Security-message-in-transit</mechanism>
      <state>n0</state>
      <inputs>
        <input>
          <key>host</key>
          <value>172.25.27.11</value>
          <domain>d1</domain>
        </input>
      </inputs>
      <expectedOutput>
        <output>
          <key>Result</key>
          <value>true</value>
          <domain>d1</domain>
        </output>
      </expectedOutput>
    </evidence>
    <evidence>
      <mechanism>encryption-internal-communications-HTTPS</mechanism>
      <state>n1</state>
      <inputs>
        <input>
          <key>serviceUri</key>
          <value>internalService.uri</value>
          <domain>d1</domain>
        </input>
```

```
        </inputs>
        <expectedOutput>
          <output>
            <key>Result</key>
            <value>true</value>
            <domain>d1</domain>
          </output>
        </expectedOutput>
      </evidence>
      <evidence>
        <mechanism>encryption-at-rest-encrypted-FS</mechanism>
        <state>n2</state>
        <inputs>
          <input>
            <key>local</key>
            <value>/dev/sda1</value>
            <domain>d1</domain>
          </input>
        </inputs>
        <expectedOutput>
          <output>
            <key>Result</key>
            <value>true</value>
            <domain>d1</domain>
          </output>
        </expectedOutput>
      </evidence>
    </evidences>
</allEvidences>
```

B.2 Code

B.3 Heuristics

We propose two heuristics balancing efficiency and quality in terms of precision and recall.

B.3.1 Heuristic 1

Psuedo-Code k-match:

```
/*****************************************************************************/
/* ALGO_SPACE is a costant corresponding to k, which is the number of possible
/* solutions to manage.
/*
/* instancep is an array containing the CM instance flows
/* templatep is an array containing the CM Template flows
```

```
7  /*
8  /* tocheckT is the index of the CM Tempalte flow under comparison.
9  /* k-match heuristic uses as pivot array tamplatep, indeed it checks once every
10 /* CM Tempate flows against all CM Instance flows.
11 /* checked is the array of CM instance flows already mapped in CM Template flows.
12 /*
13 /* heuristic starts with k-match(instancep,tempaltep,0,[])
14 /*****************************************************************************/
15 procedure k-match(path[] instancep,path[] templatep,int tocheckT, path[] checked):
16     path[][] solutions:=[][]
17     int k:=ALGO_SPACE
18     len_templatep=length of templatep
19     len_instancep=length of instancep
20     if (tocheckT <= len_templatep) then
21         for (i=1;i<=len_instancep;i++):
22             if (instancep[i] is_not_in  checked ) then
23                 c:=compatibility(tempaltep[tocheckT],instancep[i])
24                 if c == True then
25                     if k <> 0 then
26                         checked_app=checked
27                         checked_app.append(instancep[i])
28                         k-match(instancep,templatep,tocheckT+1,checked_app)
29                         k=k-1
30                     else
31                         break;
32                     endif
33                 endif
34             endif
35         endfor
36
37     else
38         solutions.append(checked)
39     endif
40     return solutions
41 end_procedure
```

B.3.2 Heuristic 2

```
1 /*****************************************************************************/
2 /* it uses the same function k-match, but instancep and templatep are ordered
3 /* based on the hierarchy of mechanisms HM
4 /* ordered(path[] paths) is the ordering function, returns an ordered array
5
6 solutions=k-match(ordered(instancep),ordered(templatep),0,[])
```

Appendix C

Composition Cloud Experimental Results

This appendix presents additional information for Chapter 4.

C.1 Fitting Profile

Plots of heuristic-1 on all dataset (1-10) using the fitting profile.

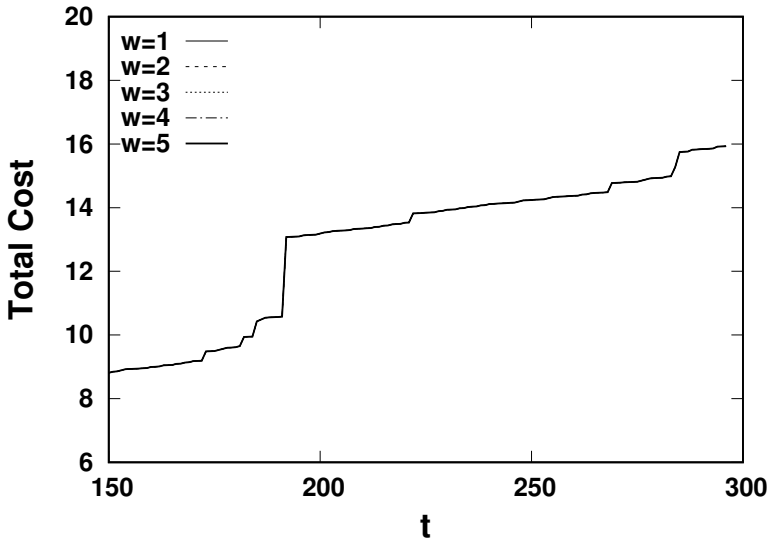


Fig. C.1 Fitting heuristic-1 dataset 1

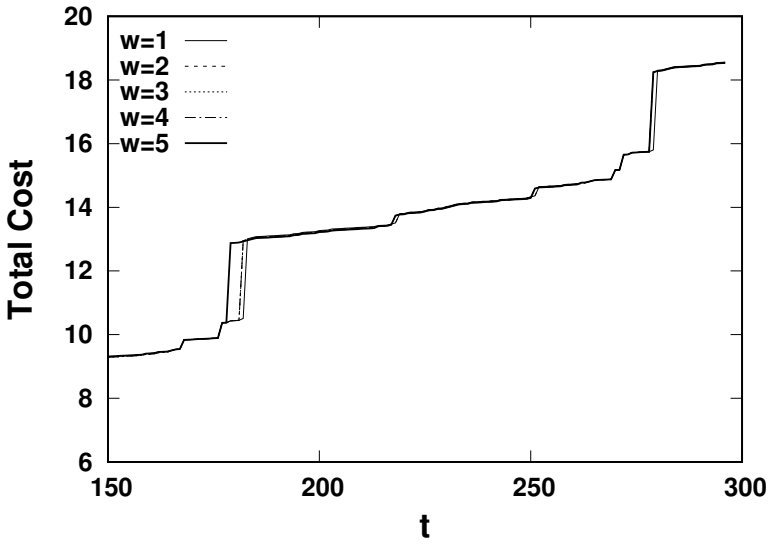


Fig. C.2 Fitting heuristic-1 dataset 2

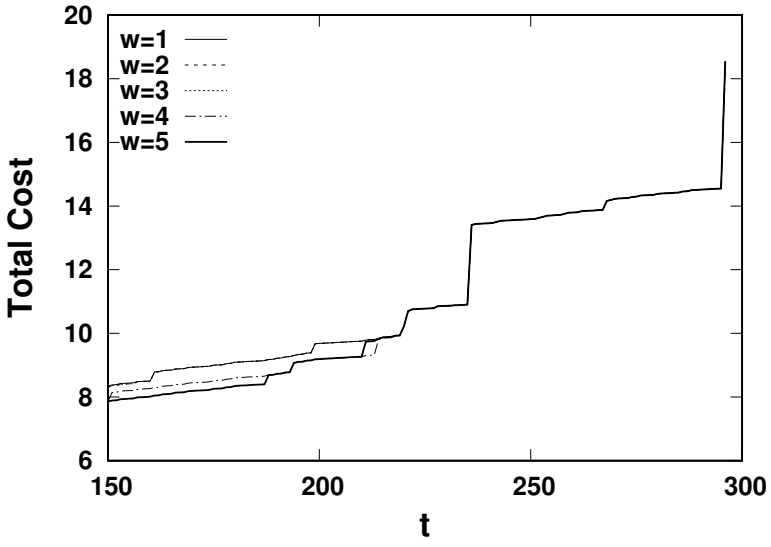


Fig. C.3 Fitting heuristic-1 dataset 3

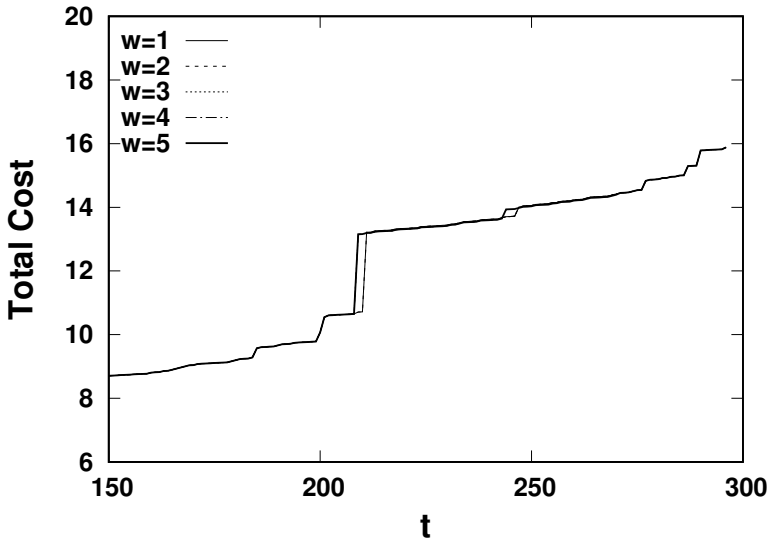


Fig. C.4 Fitting heuristic-1 dataset 4

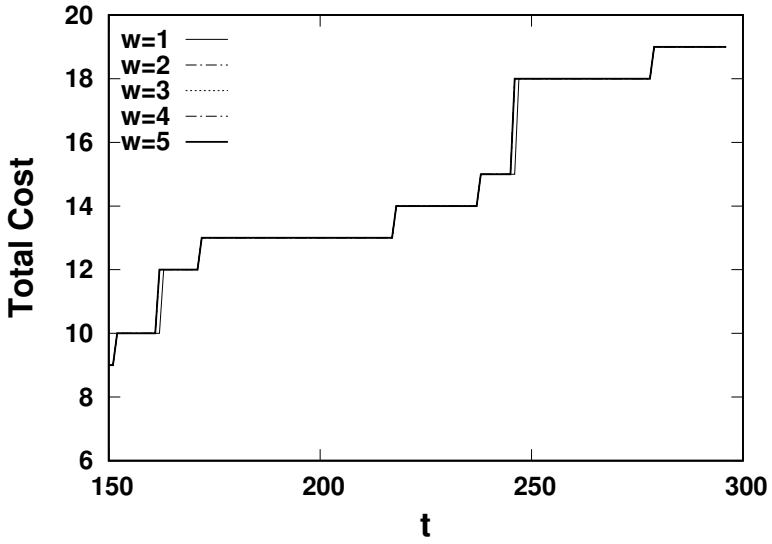


Fig. C.5 Fitting heuristic-1 dataset 5

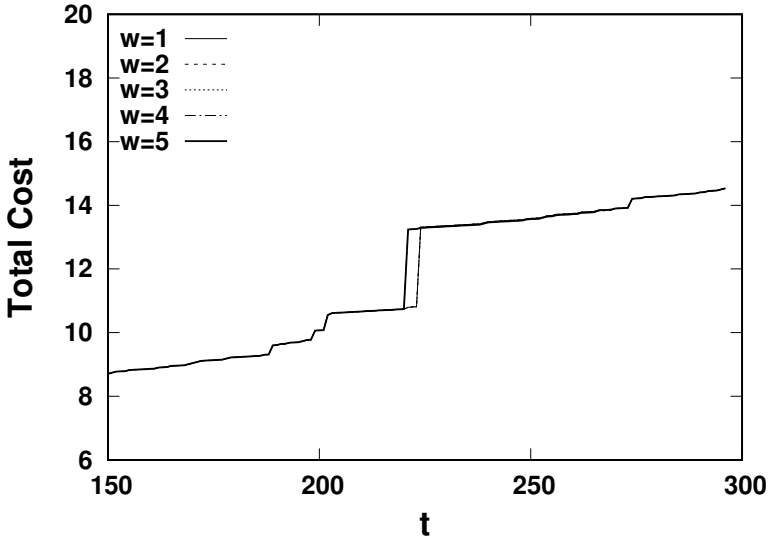


Fig. C.6 Fitting heuristic-1 dataset 6

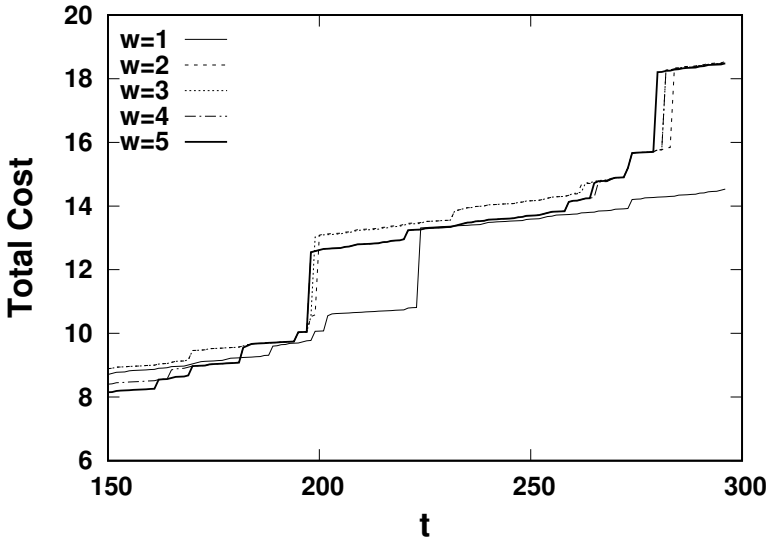


Fig. C.7 Fitting heuristic-1 dataset 7

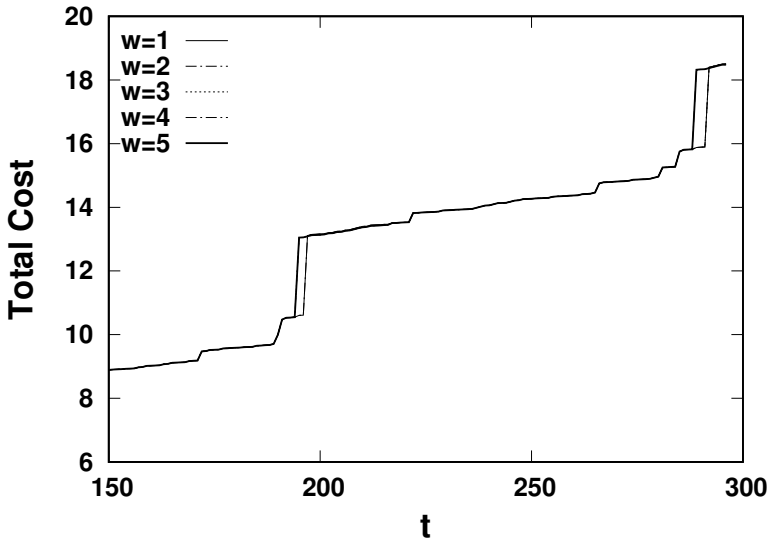


Fig. C.8 Fitting heuristic-1 dataset 8

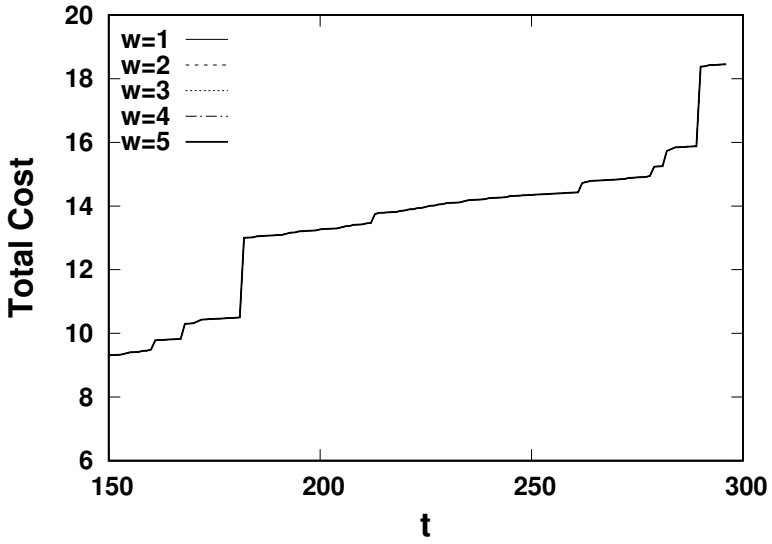


Fig. C.9 Fitting heuristic-1 dataset 9

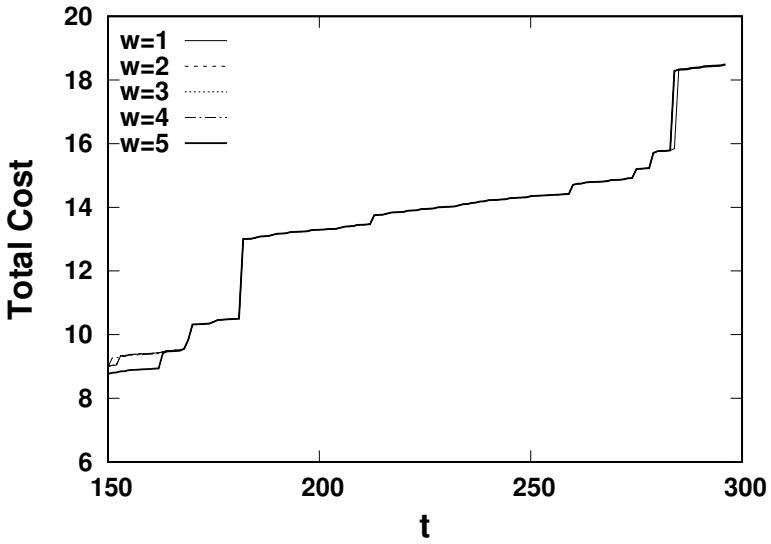


Fig. C.10 Fitting heuristic-1 dataset 10

C.2 Sharing Profile

Plots of heuristic-1 on all dataset (1-10) using the sharing profile.

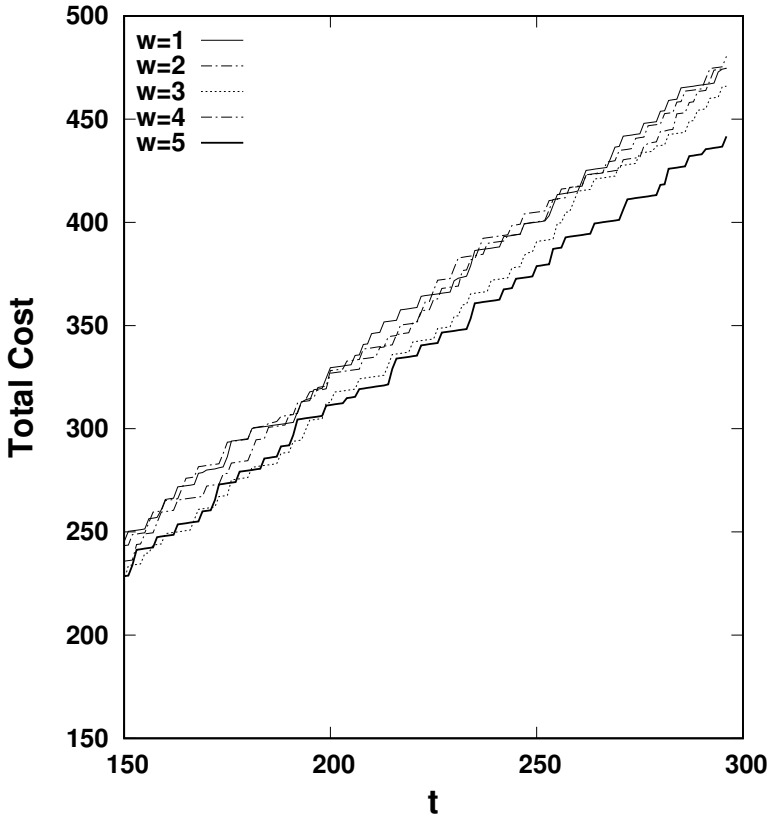


Fig. C.11 Sharing heuristic-1 dataset 1

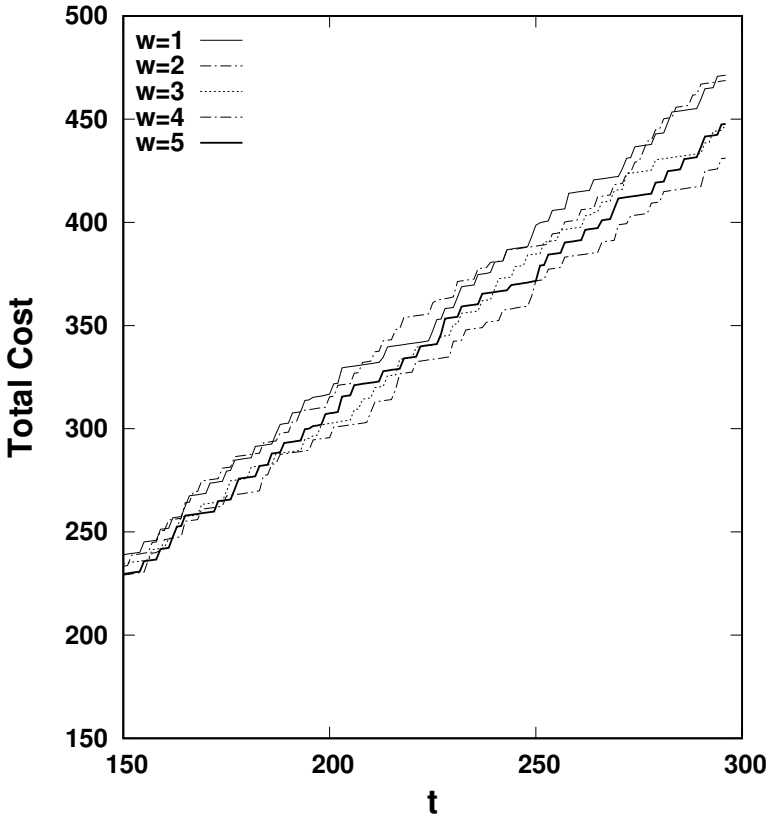


Fig. C.12 Sharing heuristic-1 dataset 2

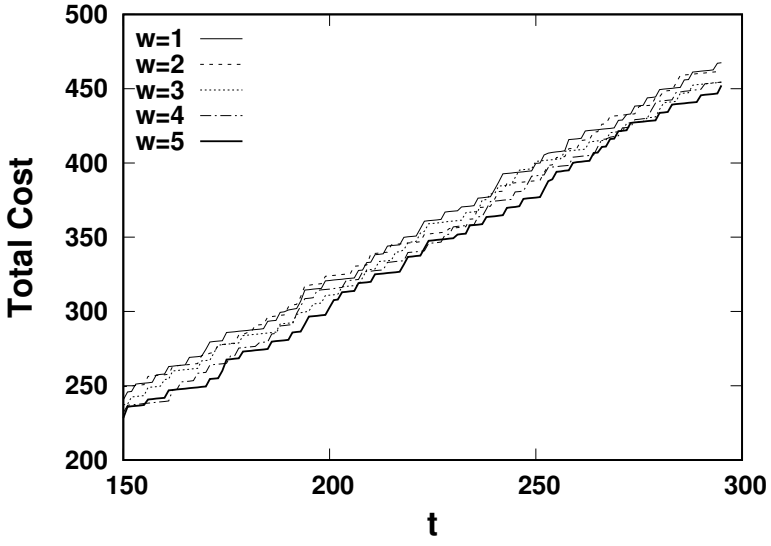


Fig. C.13 Sharing heuristic-1 dataset 3

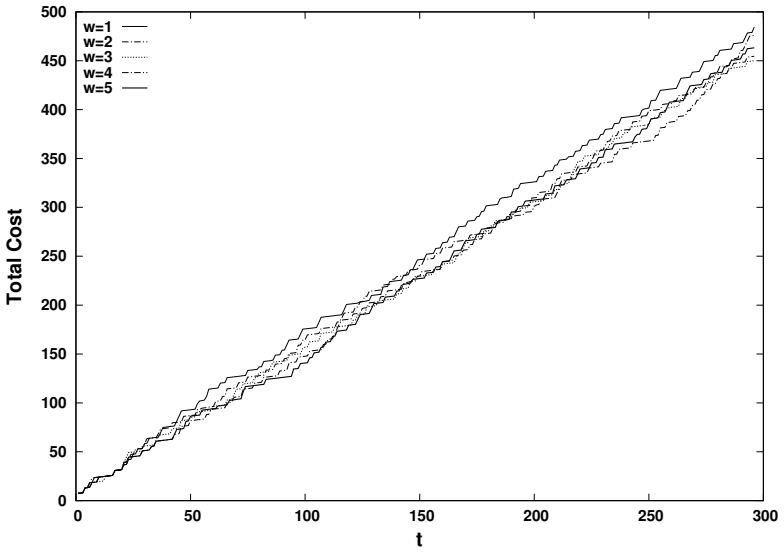


Fig. C.14 Sharing heuristic-1 dataset 4

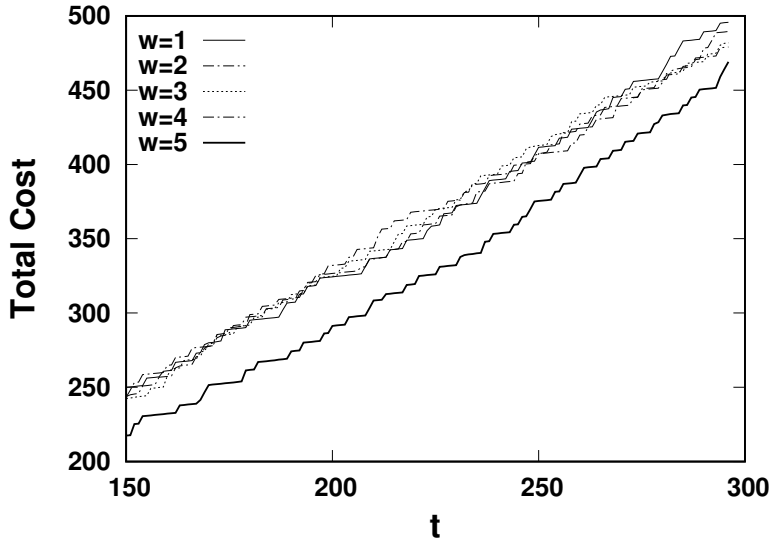


Fig. C.15 Sharing heuristic-1 dataset 5

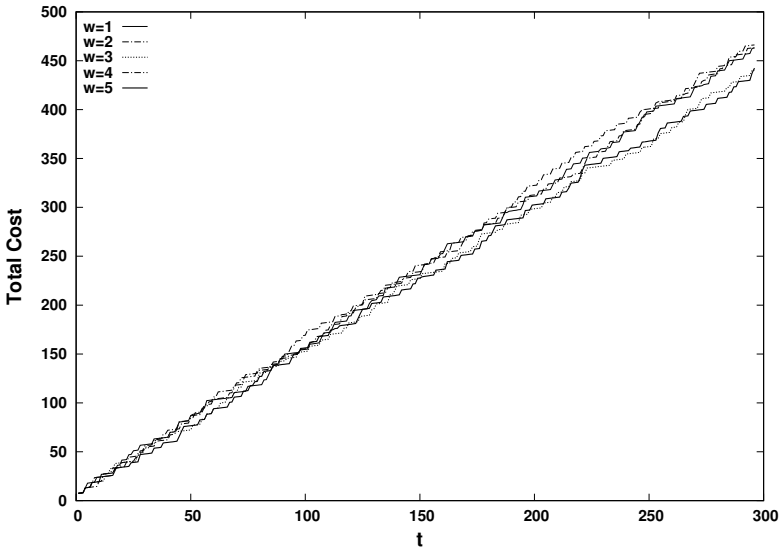


Fig. C.16 Sharing heuristic-1 dataset 6

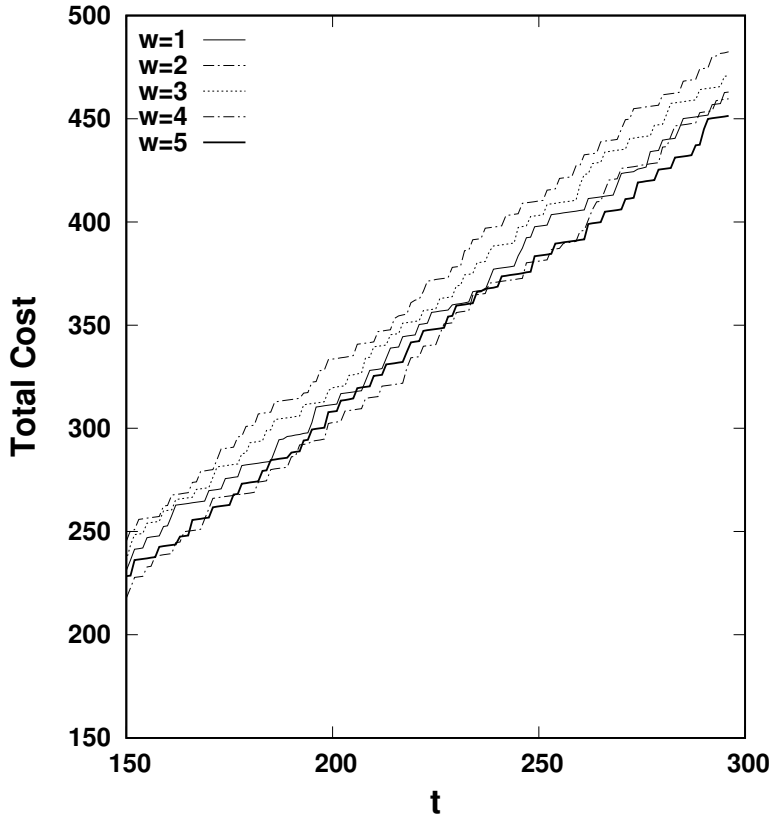


Fig. C.17 Sharing heuristic-1 dataset 7

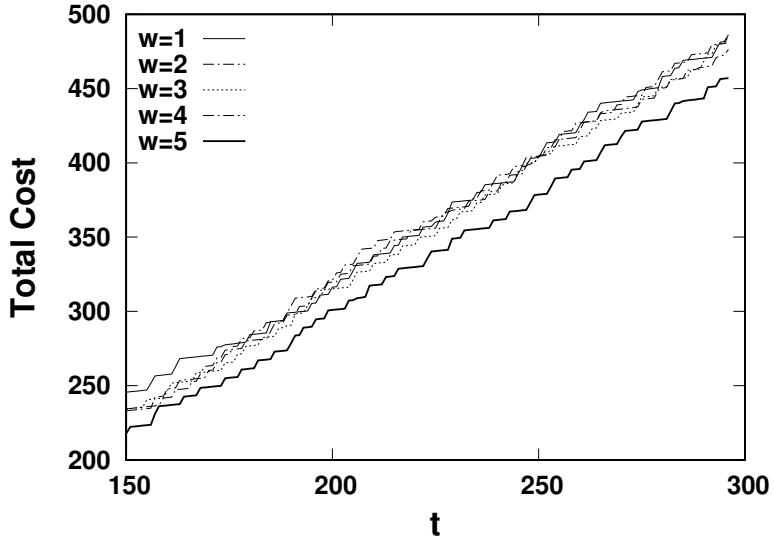


Fig. C.18 Sharing heuristic-1 dataset 8

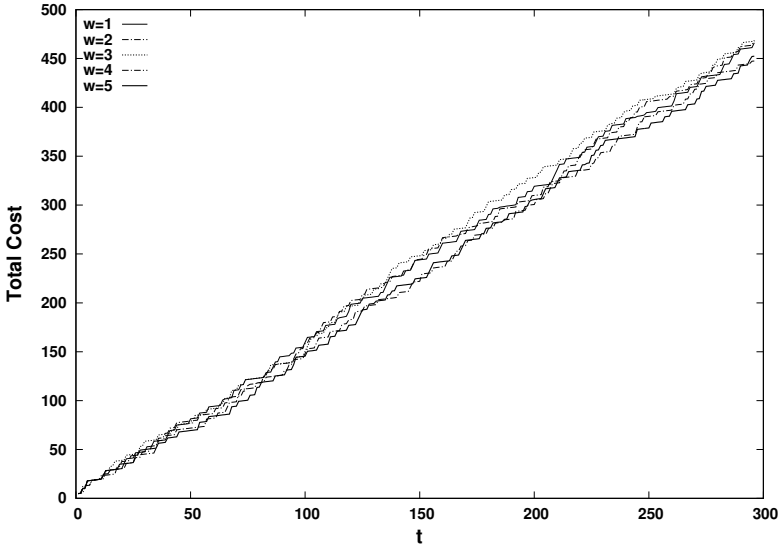


Fig. C.19 Sharing heuristic-1 dataset 9

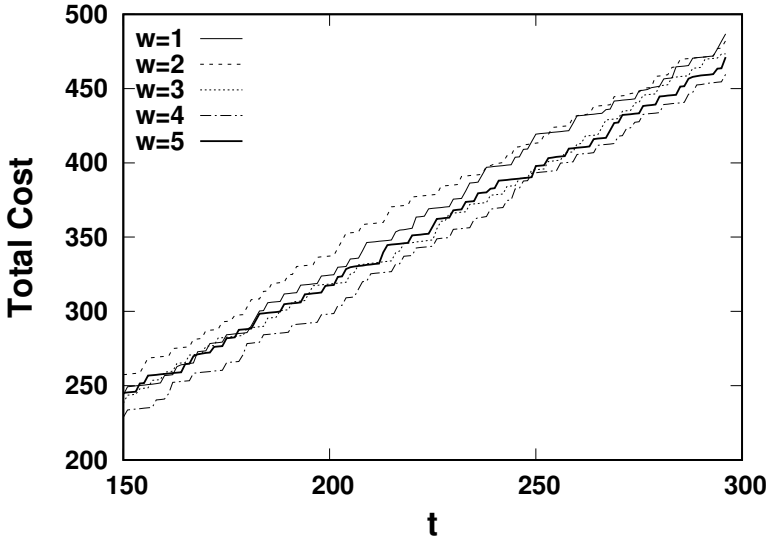


Fig. C.20 Sharing heuristic-1 dataset 10

Appendix D

Probes - OpenStack Scenario

D.1 Maintain Time Synchronization Services

ToE: All nodes that compose the OpenStack deployment. Control: The control needs to access every node and checks if the time synchronisation is enabled and if it is connected to the same server list as required. The control supports both crony and ntp.

1. connect_to_server [username, password, private_key, private_key_passphrase,hostname, port]: control accesses trough ssh the node;
2. check_timesync_enabled [ntp,chrony]: control checks, using the init system, if crony or ntp is enabled;
3. check_timesync_config [ntp_config_file (optional),chrony_config_file (optional),servers_list]: control checks that servers list in the crony or ntp config file are the same as passed in the parameters.

The Environmental settings

- Control must be executed with access to the internal network
- The paramiko python library to let the control access through ssh.

Schema:

```
{
    "connect_to_server":{
        "type":"object",
        "title": "Server connection",
        "properties":{
            "hostname":{
                "type":"string",
                "title":"Target hostname"
            },
```

```
            "port":{
                "type":"number",
                "title":"Target port"
             },
            "username":{
                "type":"string",
                "title":"Username for SSH connection"
            },
            "password":{
                "type":"string",
                "title":"Password for SSH connection"
            },
            "private_key":{
                "type":"string",
                "title":"Private RSA key for SSH connection"
            },
            "private_key_passphrase":{
                "type":"string",
                "title":"Private RSA key passphrase"
            }
        }
    },
    "check_timesync_enabled":{
        "type":"object",
        "title": "Time synchronization daemon",
        "properties":{
            "check_ntp": {
                "type": "boolean",
                "title": "Check NTPd"
            },
            "check_chrony": {
                "type": "boolean",
                "title": "Check Chrony"
            }
        }
    },
    "check_timesync_config":{
        "type":"object",
        "title": "Daemon configuration",
        "properties":{
            "servers_list": {
                "type":"array",
                "items": {
                    "type": "string",
                    "title": "Server address"
                },
                "title": "Time synchronization servers"
            },
            "ntp_config_file": {
                "type": "string",
                "title": "NTPd config file path"
            },
            "chrony_config_file": {
                "type": "string",
                "title": "Chrony config file path"
```

```
            }
        }
    }
}
```

Parameter:

```
{
    "connect_to_server": {
        "username": "root",
        "password": "password",
        "hostname": "127.0.0.1",
        "port": 22
    },
    "check_timesync_enabled": {
        "ntp": true,
        "chrony": true
    },
    "check_timesync_config": {
        "ntp_config_file": null,
        "chrony_config_file": null,
        "servers_list": [
            "0.ubuntu.pool.ntp.org",
            "1.ubuntu.pool.ntp.org",
            "2.ubuntu.pool.ntp.org",
            "3.ubuntu.pool.ntp.org",
            "ntp.ubuntu.com"
        ]
    }
}
```

Code:

```
1  import paramiko
2  import StringIO
3  from driver import Driver
4
5
6  class SSHClient(object):
7      def ssh_connect(self, hostname, port, username, password=None, private_key=None,
           private_key_passphrase=None):
8          ssh_client = paramiko.SSHClient()
9          ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
10         if private_key:
11             private_key = paramiko.RSAKey.from_private_key(private_key, password=private_key_passphrase)
12         ssh_client.connect(hostname, port, username=username, password=password, pkey=private_key)
13         return ssh_client
14
15
16 class NtpControl(Driver,
17                  SSHClient):
18     def check_init_system(self, init_system):
19         """
```

```
20          Returns the command to check whether init process is systemd, upstart or system 5 init
21          """
22          if init_system == "systemd":
23              return """ bash -c "[[ `systemctl` =~ -\.mount ]] && echo 1 || echo 0 " """
24          elif init_system == "upstart":
25              return """ bash -c "[[ `/sbin/init --version` =~ upstart ]] && echo 1 || echo 0 " """
26          elif init_system == "sysvinit":
27              return """ bash -c "[[ -f /sbin/init && ! -L /sbin/init ]] && echo 1 || echo 0 " """
28          return None
29
30      def check_daemon_enabled(self, init_system, daemon):
31          """
32          If the init system is systemd, then returns systemctl is-enabled
33          Else if the init system is upstart or sysvinit, then grabs the service status from service --status-all
34          """
35          if init_system == "systemd":
36              return """ bash -c "systemctl is-enabled {}" """.format(daemon)
37          elif init_system == "upstart" or init_system == "sysvinit":
38              return """ bash -c "[[ $(/usr/sbin/service --status-all 2>1 | awk '/%s$/ {print $2}') == "+" ]] &&
                    echo 1 || echo 0" """ % daemon
39          return None
40
41      def check_service(self, connection, init_system, daemon):
42          command = self.check_daemon_enabled(init_system, daemon)
43          _stdin, _stdout, _stderr = connection.exec_command(command)
44          out = _stdout.readlines()
45          return len(out) > 0 and out[0].strip() == "1"
46
47      def retrieve_time_servers_cmd(self, daemon="ntp", config_file=None):
48          if not config_file:
49              if daemon == "ntp":
50                  config_file = "/etc/ntp.conf"
51              elif daemon == "chrony":
52                  config_file = "/etc/chrony.conf"
53
54          return """ awk '/^server/ {print $2}' %s """ % config_file
55
56      def retrieve_time_servers(self, connection, daemon="ntp", config_file=None):
57          command = self.retrieve_time_servers_cmd(daemon, config_file)
58          _stdin, _stdout, _stderr = connection.exec_command(command)
59          out = _stdout.readlines()
60
61          for line, s in enumerate(out):
62              out[line] = str(s).strip()
63          return out
64
65      # Step 1 - Verify Preconditions
66      def preconditions(self, inputs):
67          # Verifies that the user has provided the list of the expected time servers
68
69          expected_servers_list = []
70          expected_ti = self.testinstances.get("check_timesync_config", None)
71          if expected_ti is not None:
72              expected_servers_list = expected_ti.get("servers_list")
73          assert len(expected_servers_list) > 0
```

```
74          self.expected_servers_list = expected_servers_list
75
76          # Decides whether to check ntp or chrony. If no preference is provided in the config file, checks both of
                them
77          check_ntp = False
78          check_chrony = False
79
80          # Grabs the testinstance for ntp settings
81          ntp_settings_ti = self.testinstances.get("check_timesync_enabled", None)
82          # Grabs the value of "check" for ntp
83          if ntp_settings_ti is not None:
84              check_ntp = ntp_settings_ti.get("check_ntp", False)
85
86          # Grabs the testinstance for chrony settings
87          chrony_settings_ti = self.testinstances.get("check_timesync_enabled", None)
88          # Grabs the value of "check" for chrony
89          if chrony_settings_ti is not None:
90              check_chrony = chrony_settings_ti.get("check_chrony", False)
91
92          # If both are false, it checks both.
93          if check_ntp == False and check_chrony == False:
94              check_ntp = True
95              check_chrony = True
96
97          self.check_ntp = check_ntp or False
98          self.check_chrony = check_chrony or False
99          return True
100
101     # Step 2 - Connects to the remote server
102     def connect_to_server(self, inputs):
103         assert inputs is True
104         ssh_connection_ti = self.testinstances.get("connect_to_server", None)
105         assert not ssh_connection_ti is None
106
107         hostname = ssh_connection_ti.get("hostname")
108         port = ssh_connection_ti.get("port")
109         username = ssh_connection_ti.get("username")
110         password = ssh_connection_ti.get("password", None)
111         private_key = ssh_connection_ti.get("private_key", None)
112         if private_key is not None:
113             private_key = StringIO.StringIO(private_key)
114             private_key_passphrase = ssh_connection_ti.get("private_key_passphrase", None)
115
116         assert not password is None or not private_key is None
117
118
119         self.ssh_connection = self.ssh_connect(
120             hostname=hostname,
121             username=username,
122             port=port,
123             password=password,
124             private_key=private_key,
125             private_key_passphrase=private_key_passphrase
126             )
127         return True
```

```
128
129     # Step 3 - Identifies the init system used by the remote server
130     def identify_init_system(self, inputs):
131         assert inputs is True
132
133         _stdin, _stdout, _stderr = self.ssh_connection.exec_command(self.check_init_system("systemd"))
134         out = _stdout.readlines()
135         if len(out) > 0 and out[0].strip() == "1":
136             return "systemd"
137
138         """
139         Checks if init system is upstart
140         """
141         _stdin, _stdout, _stderr = self.ssh_connection.exec_command(self.check_init_system("upstart"))
142         out = _stdout.readlines()
143
144         if len(out) > 0 and out[0].strip() == "1":
145             return "upstart"
146
147         """
148         Checks if init system is sysvinit
149         """
150         _stdin, _stdout, _stderr = self.ssh_connection.exec_command(self.check_init_system("sysvinit"))
151         out = _stdout.readlines()
152         if len(out) > 0 and out[0].strip() == "1":
153             return "sysvinit"
154
155     # Step 4 - Verifies that the init system is one of the supported ones (step2 returns a valid value)
156     def verify_init_system(self, init_system):
157         assert init_system in ("sysvinit", "systemd", "upstart", )
158         return init_system
159
160     def check_ntp_enabled(self, init_system):
161         self.has_ntp = False
162         if self.check_ntp:
163             self.has_ntp = self.check_service(self.ssh_connection, init_system, "ntp")
164         return init_system
165
166     def check_chrony_enabled(self, init_system):
167         self.has_chrony = False
168         if self.check_chrony:
169             self.has_chrony = self.check_service(self.ssh_connection, init_system, "chrony")
170
171     def assert_ntp_or_chrony(self, inputs):
172         assert self.has_chrony ^ self.has_ntp
173         return True
174
175     # Step 5 - Checks that at least one timesync service is enabled
176
177     def check_timesync_enabled(self, init_system):
178         self.check_ntp_enabled(init_system)
179         self.check_chrony_enabled(init_system)
180         return self.assert_ntp_or_chrony(None)
181
182
```

```
183     def check_ntp_config(self, inputs):
184         if self.has_ntp:
185             ntp_settings_ti = self.testinstances.get("check_timesync_config", None)
186             if ntp_settings_ti is not None:
187                 config_file = ntp_settings_ti.get("ntp_config_file", None)
188                 servers = self.retrieve_time_servers(self.ssh_connection, daemon="ntp", config_file=config_file)
189                 print(servers)
190                 return inputs and len(set(servers) & set(self.expected_servers_list)) == len(set(servers))
191 
192         return inputs
193 
194     def check_chrony_config(self, inputs):
195         if self.has_chrony:
196             chrony_settings_ti = self.testinstances.get("check_timesync_config", None)
197             if chrony_settings_ti is not None:
198                 config_file = chrony_settings_ti.get("chrony_config_file", None)
199                 servers = self.retrieve_time_servers(self.ssh_connection , daemon="chrony", config_file=config_file
                        )
200                 return inputs and len(set(servers) & set(self.expected_servers_list)) == len(set(servers))
201         return inputs
202 
203     # Step 6 - Checks configurations for timesync services and compares the list with the provided one
204     def check_timesync_config(self, inputs):
205         ntp_config = self.check_ntp_config(inputs)
206         return self.check_chrony_config(ntp_config)
207 
208     # Step 7 - Closes connection
209     def close_ssh_connection(self, inputs):
210         try:
211             self.ssh_connection.close()
212         except:
213             pass
214         return inputs
215 
216 
217     def appendAtomics(self):
218         self.appendAtomic(self.preconditions, lambda:None)
219         self.appendAtomic(self.connect_to_server, self.close_ssh_connection)
220         self.appendAtomic(self.identify_init_system, lambda: None)
221         self.appendAtomic(self.verify_init_system, lambda: None)
222         self.appendAtomic(self.check_timesync_enabled, lambda: None)
223         self.appendAtomic(self.check_timesync_config, lambda: None)
224         self.appendAtomic(self.close_ssh_connection, lambda: None)
```

D.2 Do Not Use or Set Guest Customization Passwords for the User Profile

Profile: Cloud.

ToE: Openstack Keystone. Keystone is the identity service and manages projects, users and groups. Control Admin can't be member of any projects, excepts her owns projects and

can't change users passords. Hence, the control is double: i) admin user is only member of a restricted list of project as specified in a list. The openstack policy doens't allows to change user password and that should be changed only through the centralized identity system.

1. openstack_connection [os_username, os_password, os_project_id, os_auth_url, os_user_domain_name] using the admin credentials, control connects to OpenStack API
2. checkProject [project_list]: control parses all projects and control admin is member only of the passed projects.

The Environmental settings

- Control must be executed with access to the internal network
- The paramiko python library to let the control access through ssh.

Schema:

Parameters:

Code:

```
1
2  from novaclient.client import Client as NovaClient
3  from keystoneauth1.identity import v3 as KeystoneClient
4  from keystoneauth1 import session as KeystoneSession
5  from keystoneclient.v3 import client
6  from driver import Driver
7
8
9
10
11
12
13 class members(Driver):
14     def openstackConfig(self, inputs):
15         self.keystonecl = KeystoneClient.Password(auth_url=self.testinstances["openstackConfig"]["
               OS_AUTH_URL"],
16                         username=self.testinstances["openstackConfig"]["OS_USERNAME"],
17                         password=self.testinstances["openstackConfig"]["OS_PASSWORD"],
18                         project_id=self.testinstances["openstackConfig"]["OS_PROJECT_ID"],
19                         user_domain_name=self.testinstances["openstackConfig"]["OS_USER_DOMAIN_NAME
                               "])
20         sess = KeystoneSession.Session(auth=self.keystonecl)
21         keystone = client.Client(session=sess)
22         return keystone
23
24     def checkProject(self,keystone):
```

```
25          elem=self.testinstances["checkProject"]
26          members=elem.get("members")
27          project=elem.get("project")
28          users=keystone.users.list(default_project=project)
29          m=0
30          for u in users:
31              projects = keystone.projects.list(user=u.id)
32              for p in projects:
33                  if p.name == project:
34                      if not u.id in members:
35                          return False
36                      else:
37                          m=m+1
38          if m == len(members):
39              return True
40          else:
41              return False
42
43
44      def appendAtomics(self):
45          self.appendAtomic(self.openstackConfig, lambda:None)
46          self.appendAtomic(self.checkProject,lambda:None)
```

The execution flow of control ii) consists of two sequen- tial operations with the relative Parameters as follows.

1. connect_to_server [username, password, private_key, private_key_passphrase, hostname, port]: control accesses through ssh the Keystone nodes.
2. retrieve_policy_file[path]:controls reads and parses the policy file.
3. inspect_policy_file [key, expected_value]: control checks that identity:change_password action is disabled.

The Environmental settings are the following:

- Control must be executed with access to the internal network.
- The paramiko python library to let the control access through ssh.

Schema:

```
{
    "connect_to_server":{
        "type":"object",
        "title": "Server connection",
        "properties":{
            "hostname":{
                "type":"string",
                "title":"Target hostname"
```

```
            },
            "port":{
                "type":"number",
                "title":"Target port"
             },
            "username":{
                "type":"string",
                "title":"Username for SSH connection"
            },
            "password":{
                "type":"string",
                "title":"Password for SSH connection"
            },
            "private_key":{
                "type":"string",
                "title":"Private RSA key for SSH connection"
            },
            "private_key_passphrase":{
                "type":"string",
                "title":"Private RSA key passphrase"
            }
        }
    },
    "retrieve_policy_file":{
        "type":"object",
        "title": "Policy file",
        "properties":{
            "path": {
                "type": "string",
                "title": "File path"
            }
        }
    },
    "inspect_policy_file":{
        "type":"object",
        "title": "Inspection",
        "properties":{
            "key": {
                "type": "string",
                "title": "Policy name"
            },
            "expected_value": {
                "type":"",
                "title": "Expected value"
            }
        }
    }
}
```

Parameter:

```
{
    "connect_to_server": {
        "username": "root",
```

```
        "password": "password",
        "hostname": "127.0.0.1",
        "port": 22
    },
    "retrieve_policy_file": {
        "path": "/etc/keystone/policy.json"
    },
    "inspect_policy_file": {
        "key": "identity:change_password",
        "expected_value": "!"
    }
}
```

Code:

```
1  import paramiko
2  import StringIO
3  from driver import Driver
4  import json
5
6  class SSHClient(object):
7      def ssh_connect(self, hostname, port, username, password=None, private_key=None,
           private_key_passphrase=None):
8          ssh_client = paramiko.SSHClient()
9          ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
10         if private_key:
11             private_key = paramiko.RSAKey.from_private_key(private_key, password=private_key_passphrase)
12         ssh_client.connect(hostname, port, username=username, password=password, pkey=private_key)
13         return ssh_client
14
15
16 class OpenstackPolicyControl(Driver,
17                       SSHClient):
18
19     policy_file = None
20     policy_key = None
21     policy_expected_value = None
22
23     # Step 1 - Check prerequisites
24     def prerequisites(self, inputs):
25         retrieve_policy_configuration_ti = self.testinstances.get("retrieve_policy_file", None)
26         if retrieve_policy_configuration_ti is not None:
27             self.policy_file = retrieve_policy_configuration_ti.get("path", None)
28
29         expected_ti = self.testinstances.get("inspect_policy_file", None)
30         if expected_ti is not None:
31             self.policy_key = expected_ti.get("key", None)
32             self.policy_expected_value = expected_ti.get("expected_value", None)
33
34         assert self.policy_file is not None and self.policy_key is not None
35         return True
36
37     # Step 2 - Connects to the remote server
38     def connect_to_server(self, inputs):
```

```
39          assert inputs is True
40          ssh_connection_ti = self.testinstances.get("connect_to_server", None)
41
42          assert not ssh_connection_ti is None
43
44          hostname = ssh_connection_ti.get("hostname")
45          port = ssh_connection_ti.get("port")
46          username = ssh_connection_ti.get("username")
47          password = ssh_connection_ti.get("password", None)
48          private_key = ssh_connection_ti.get("private_key", None)
49          if private_key is not None:
50              private_key = StringIO.StringIO(private_key)
51              private_key_passphrase = ssh_connection_ti.get("private_key_passphrase", None)
52
53          assert not password is None or not private_key is None
54
55
56          self.ssh_connection = self.ssh_connect(
57              hostname=hostname,
58              username=username,
59              port=port,
60              password=password,
61              private_key=private_key,
62              private_key_passphrase=private_key_passphrase
63              )
64          return True
65
66      # Retrieves keystone configuration from remote server (OpenStack controller)
67      def retrieve_policy_file(self, inputs):
68          assert self.policy_file is not None
69          _stdin, _stdout, _stderr = self.ssh_connection.exec_command("cat %s" % self.policy_file)
70          lines = _stdout.readlines()
71          return lines
72
73      def inspect_policy_file(self, lines):
74          policies = '\n'.join(lines)
75          parsed_policies = json.loads(policies)
76          return parsed_policies.get(self.policy_key) == self.policy_expected_value
77
78      def close_ssh_connection(self, inputs):
79          try:
80              self.ssh_connection.close()
81          except:
82              pass
83          return inputs
84
85
86      def appendAtomics(self):
87          self.appendAtomic(self.prerequisites, lambda: None)
88          self.appendAtomic(self.connect_to_server, self.close_ssh_connection)
89          self.appendAtomic(self.retrieve_policy_file, lambda: None)
90          self.appendAtomic(self.inspect_policy_file, lambda: None)
91          self.appendAtomic(self.close_ssh_connection, lambda: None)
```

D.3 Evaluate Cloud Architecture Dependencies

Profile: User.

ToE: Nova computing and user VMs Control User can mitigate the dependencies from single point of failure of a cloud by deploying her VMs in different availability zone; hence, the control checks that a set of VM are at least deployed in two different availability zone.

The execution flow of the first Controls C1 consists of three sequential operations with the relative Parameters as follows.

1. openstack-connection [user credentials]: using user credentials to access OpenStack API
2. retrieve-zone []: control retrieves all availability zones in OpenStack.
3. check-deployment[vm-list]:controlchecksthatatleast one VM from vm-list is deployed in a different availability zone.

The Environmental settings

- Control must be executed with access to the public OpenStack API.
- The OpenStack client SDK to be able to communicate with its API.

Schema:

```
{
    "connect_to_server":{
        "type":"object",
        "title": "Server connection",
        "properties":{
            "hostname":{
                "type":"string",
                "title":"Target hostname"
            },
            "port":{
                "type":"number",
                "title":"Target port"
             },
            "username":{
                "type":"string",
                "title":"Username for SSH connection"
            },
            "password":{
                "type":"string",
                "title":"Password for SSH connection"
            },
            "private_key":{
                "type":"string",
                "title":"Private RSA key for SSH connection"
            },
            "private_key_passphrase":{
                "type":"string",
```

```
                "title":"Private RSA key passphrase"
            }
        }
    },
    "retrieve_services_configurations":{
        "type":"object",
        "title": "Services configuration",
        "properties":{
            "nova_config_file": {
                "type": "string",
                "title": "Nova configuration file path"
            },
            "cinder_config_file": {
                "type":"string",
                "title":"Cinder configuration file path"
            }
        }
    }
}
```

Parameters:

```
{
    "connect_to_server": {
        "username": "root",
        "password": "password",
        "hostname": "127.0.0.1",
        "port": 22
    },
    "retrieve_services_configurations": {
        "nova_config_file": null,
        "cinder_config_file": null
    }
}
```

Code:

```
1  import paramiko
2  import re
3  import StringIO
4  from driver import Driver
5  from oslo_config import cfg
6
7  class SSHClient(object):
8      def ssh_connect(self, hostname, port, username, password=None, private_key=None,
           private_key_passphrase=None):
9          ssh_client = paramiko.SSHClient()
10         ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
11         if private_key:
12             private_key = paramiko.RSAKey.from_private_key(private_key, password=private_key_passphrase)
13         ssh_client.connect(hostname, port, username=username, password=password, pkey=private_key)
14         return ssh_client
15
```

```
16 class MyConfigParser(cfg.ConfigParser):
17     def ___init___(self, filename, ldap_config):
18         super(cfg.ConfigParser, self).___init___()
19         self.sections = {}
20         self._normalized = None
21         self.section = None
22         self.ldap_config = ldap_config
23         self.filename = filename
24 
25     def parse(self):
26         return super(cfg.ConfigParser, self).parse(self.ldap_config)
27 
28 class CinderNovaEncryptedFixedKey(Driver,
29                         SSHClient):
30 
31     keystone_config_file = None
32     # Step 1 - Check prerequisites
33     def prerequisites(self, inputs):
34         self.nova_config_file = "/etc/nova/nova.conf"
35         self.cinder_config_file = "/etc/cinder/cinder.conf"
36 
37         retrieve_services_configurations_ti = self.testinstances.get("retrieve_services_configurations", None)
38 
39         if retrieve_services_configurations_ti is not None:
40             self.nova_config_file = retrieve_nova_configuration_ti.get("nova_config_path", self.nova_config_file)
41             self.cinder_config_file = retrieve_cinder_configuration_ti.get("cinder_config_path", self.
                    cinder_config_file)
42 
43         return True
44 
45     # Step 2 - Connects to the remote server
46     def connect_to_server(self, inputs):
47         assert inputs is True
48         ssh_connection_ti = self.testinstances.get("connect_to_server", None)
49 
50         assert not ssh_connection_ti is None
51 
52         hostname = ssh_connection_ti.get("hostname")
53         port = ssh_connection_ti.get("port")
54         username = ssh_connection_ti.get("username")
55         password = ssh_connection_ti.get("password", None)
56         private_key = ssh_connection_ti.get("private_key", None)
57         if private_key is not None:
58             private_key = StringIO.StringIO(private_key)
59             private_key_passphrase = ssh_connection_ti.get("private_key_passphrase", None)
60 
61         assert not password is None or not private_key is None
62 
63 
64         self.ssh_connection = self.ssh_connect(
65             hostname=hostname,
66             username=username,
67             port=port,
68             password=password,
69             private_key=private_key,
```

```
70              private_key_passphrase=private_key_passphrase
71              )
72          return True
73
74      # Retrieves nova configuration from remote server (OpenStack controller)
75      def retrieve_nova_configuration(self):
76          assert self.nova_config_file is not None
77          _stdin, _stdout, _stderr = self.ssh_connection.exec_command("cat %s" % self.nova_config_file)
78          lines = _stdout.readlines()
79          return lines
80
81      # Retrieves cinder configuration from remote server (OpenStack controller)
82      def retrieve_cinder_configuration(self):
83          assert self.cinder_config_file is not None
84          _stdin, _stdout, _stderr = self.ssh_connection.exec_command("cat %s" % self.cinder_config_file)
85          lines = _stdout.readlines()
86          return lines
87
88      def retrieve_services_configurations(self, inputs):
89          return self.retrieve_nova_configuration(), self.retrieve_cinder_configuration()
90
91      def check_strength(self, passphrase):
92          classes = {}
93          classes["alphabetic_class"] = "[A-z]"
94          classes["numeric_class"] = "[0-9]"
95          classes["symbol_class "]= "[!@#$%^&*()_\+\|\~\-=`\\\\{\}\[\]:\";'<>?,./]"
96          count = 0
97          for single_class in classes:
98              if re.search(classes[single_class], passphrase) is not None:
99                  count += 1
100         return len(passphrase) >= 9 and count >= 2
101
102     def check_fixed_key_nova(self, nova_config):
103         mcp = MyConfigParser(self.nova_config_file, nova_config)
104         mcp.parse()
105         section = mcp.sections.get("key_manager", mcp.sections.get("keymgr", None))
106         assert section is not None
107         fixed_key = section.get("fixed_key")[0]
108         assert fixed_key is not None
109         return self.check_strength(fixed_key)
110
111     def check_fixed_key_cinder(self, cinder_config):
112         mcp = MyConfigParser(self.cinder_config_file, cinder_config)
113         mcp.parse()
114         section = mcp.sections.get("key_manager", mcp.sections.get("keymgr", None))
115         assert section is not None
116         fixed_key = section.get("fixed_key")[0]
117         assert fixed_key is not None
118         return self.check_strength(fixed_key)
119
120     def check_fixed_keys(self, inputs):
121         nova_config, cinder_config = inputs
122         return self.check_fixed_key_cinder(cinder_config) and self.check_fixed_key_nova(nova_config)
123
124     def close_ssh_connection(self, inputs):
```

```
125             try:
126                 self.ssh_connection.close()
127             except:
128                 pass
129             return inputs
130
131
132         def appendAtomics(self):
133             self.appendAtomic(self.prerequisites, lambda: None)
134             self.appendAtomic(self.connect_to_server, self.close_ssh_connection)
135             self.appendAtomic(self.retrieve_services_configurations, lambda:None)
136             self.appendAtomic(self.check_fixed_keys, lambda: None)
137             self.appendAtomic(self.close_ssh_connection, lambda: None)
```

D.4 nova-cinder-encryption-fixed-key

Profile: Cloud.
ToE:

1. uno

The Environmental settings

- uno

Schema:

```
{
    "connect_to_server":{
        "type":"object",
        "title": "Server connection",
        "properties":{
            "hostname":{
                "type":"string",
                "title":"Target hostname"
            },
            "port":{
                "type":"number",
                "title":"Target port"
             },
            "username":{
                "type":"string",
                "title":"Username for SSH connection"
            },
            "password":{
                "type":"string",
                "title":"Password for SSH connection"
            },
            "private_key":{
                "type":"string",
```

```
                "title":"Private RSA key for SSH connection"
            },
            "private_key_passphrase":{
                "type":"string",
                "title":"Private RSA key passphrase"
            }
        }
    },
    "retrieve_services_configurations":{
        "type":"object",
        "title": "Services configuration",
        "properties":{
            "nova_config_file": {
                "type": "string",
                "title": "Nova configuration file path"
            },
            "cinder_config_file": {
                "type":"string",
                "title":"Cinder configuration file path"
            }
        }
    }
}
```

Parameters:

```
{
    "connect_to_server": {
    "username": "root",
    "password": "password",
    "hostname": "127.0.0.1",
    "port": 22
    },
    "retrieve_services_configurations": {
        "nova_config_file": null,
        "cinder_config_file": null
    }
}
```

Code:

```
1  import paramiko
2  import re
3  import StringIO
4  from driver import Driver
5  from oslo_config import cfg
6
7  class SSHClient(object):
8      def ssh_connect(self, hostname, port, username, password=None, private_key=None,
           private_key_passphrase=None):
9          ssh_client = paramiko.SSHClient()
10         ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
11         if private_key:
```

```
12              private_key = paramiko.RSAKey.from_private_key(private_key, password=private_key_passphrase)
13          ssh_client.connect(hostname, port, username=username, password=password, pkey=private_key)
14          return ssh_client
15
16  class MyConfigParser(cfg.ConfigParser):
17      def ___init___(self, filename, ldap_config):
18          super(cfg.ConfigParser, self).___init___()
19          self.sections = {}
20          self._normalized = None
21          self.section = None
22          self.ldap_config = ldap_config
23          self.filename = filename
24
25      def parse(self):
26          return super(cfg.ConfigParser, self).parse(self.ldap_config)
27
28  class CinderNovaEncryptedFixedKey(Driver,
29                        SSHClient):
30
31      keystone_config_file = None
32      # Step 1 - Check prerequisites
33      def prerequisites(self, inputs):
34          self.nova_config_file = "/etc/nova/nova.conf"
35          self.cinder_config_file = "/etc/cinder/cinder.conf"
36
37          retrieve_services_configurations_ti = self.testinstances.get("retrieve_services_configurations", None)
38
39          if retrieve_services_configurations_ti is not None:
40              self.nova_config_file = retrieve_nova_configuration_ti.get("nova_config_path", self.nova_config_file)
41              self.cinder_config_file = retrieve_cinder_configuration_ti.get("cinder_config_path", self.
                        cinder_config_file)
42
43          return True
44
45      # Step 2 - Connects to the remote server
46      def connect_to_server(self, inputs):
47          assert inputs is True
48          ssh_connection_ti = self.testinstances.get("connect_to_server", None)
49
50          assert not ssh_connection_ti is None
51
52          hostname = ssh_connection_ti.get("hostname")
53          port = ssh_connection_ti.get("port")
54          username = ssh_connection_ti.get("username")
55          password = ssh_connection_ti.get("password", None)
56          private_key = ssh_connection_ti.get("private_key", None)
57          if private_key is not None:
58              private_key = StringIO.StringIO(private_key)
59              private_key_passphrase = ssh_connection_ti.get("private_key_passphrase", None)
60
61          assert not password is None or not private_key is None
62
63
64          self.ssh_connection = self.ssh_connect(
65              hostname=hostname,
```

```
66              username=username,
67              port=port,
68              password=password,
69              private_key=private_key,
70              private_key_passphrase=private_key_passphrase
71              )
72          return True
73
74      # Retrieves nova configuration from remote server (OpenStack controller)
75      def retrieve_nova_configuration(self):
76          assert self.nova_config_file is not None
77          _stdin, _stdout, _stderr = self.ssh_connection.exec_command("cat %s" % self.nova_config_file)
78          lines = _stdout.readlines()
79          return lines
80
81      # Retrieves cinder configuration from remote server (OpenStack controller)
82      def retrieve_cinder_configuration(self):
83          assert self.cinder_config_file is not None
84          _stdin, _stdout, _stderr = self.ssh_connection.exec_command("cat %s" % self.cinder_config_file)
85          lines = _stdout.readlines()
86          return lines
87
88      def retrieve_services_configurations(self, inputs):
89          return self.retrieve_nova_configuration(), self.retrieve_cinder_configuration()
90
91      def check_strength(self, passphrase):
92          classes = {}
93          classes["alphabetic_class"] = "[A-z]"
94          classes["numeric_class"] = "[0-9]"
95          classes["symbol_class "]= "[!@#$%^&*()_\+\|\~\-=‘\\\\{\}\[\]:\";’<>?,./]"
96          count = 0
97          for single_class in classes:
98              if re.search(classes[single_class], passphrase) is not None:
99                  count += 1
100         return len(passphrase) >= 9 and count >= 2
101
102     def check_fixed_key_nova(self, nova_config):
103         mcp = MyConfigParser(self.nova_config_file, nova_config)
104         mcp.parse()
105         section = mcp.sections.get("key_manager", mcp.sections.get("keymgr", None))
106         assert section is not None
107         fixed_key = section.get("fixed_key")[0]
108         assert fixed_key is not None
109         return self.check_strength(fixed_key)
110
111     def check_fixed_key_cinder(self, cinder_config):
112         mcp = MyConfigParser(self.cinder_config_file, cinder_config)
113         mcp.parse()
114         section = mcp.sections.get("key_manager", mcp.sections.get("keymgr", None))
115         assert section is not None
116         fixed_key = section.get("fixed_key")[0]
117         assert fixed_key is not None
118         return self.check_strength(fixed_key)
119
120     def check_fixed_keys(self, inputs):
```

```
121         nova_config, cinder_config = inputs
122         return self.check_fixed_key_cinder(cinder_config) and self.check_fixed_key_nova(nova_config)
123
124     def close_ssh_connection(self, inputs):
125         try:
126             self.ssh_connection.close()
127         except:
128             pass
129         return inputs
130
131
132     def appendAtomics(self):
133         self.appendAtomic(self.prerequisites, lambda: None)
134         self.appendAtomic(self.connect_to_server, self.close_ssh_connection)
135         self.appendAtomic(self.retrieve_services_configurations, lambda:None)
136         self.appendAtomic(self.check_fixed_keys, lambda: None)
137         self.appendAtomic(self.close_ssh_connection, lambda: None)
```

D.5 Central Directory for Authentication and Authorization for the cloud profile

Profile: Cloud
ToE: OpenStack Keystone. We note that Keystone offers the possibility to be integrated with an external identity access management. Control: The control verify that Keystone is configured to use the company internal LDAP. The Keystone configuration is defined in every node where keystone is running. Keystone nodes can be retrieved by OpenStack API or specifying manually their IP addresses. The control needs to access to all nodes running Keystone and control that the key- stone.configuration contains all necessary key-value to be connected to the internal authentication system.

1. connect_to_server [username, password, private_key, private_key_passphrase, hostname, port]: connect ssh to the Keystone nodes.

2. retrieve_keystone_configuration [keystone_config_file(optional)]: read from the Keystone conifig placed in /etc/keystone/keystone.conf if keystone_config_file is not passed.

3. textitcheck_ldap [ldap_url]: the control check the the ldap driver is enabled and check the the required ldap is configured correctly

The Environmental settings

- The control must be executed with access to the internal network.

- The OpenStack python configuration library to be able to parse OpenStack configurations.
- The paramiko python library to let the control access through ssh.

Schema:

```
{
    "connect_to_server":{
        "type":"object",
        "title": "Server connection",
        "properties":{
            "hostname":{
                "type":"string",
                "title":"Target hostname"
            },
            "port":{
                "type":"number",
                "title":"Target port"
             },
            "username":{
                "type":"string",
                "title":"Username for SSH connection"
            },
            "password":{
                "type":"string",
                "title":"Password for SSH connection"
            },
            "private_key":{
                "type":"string",
                "title":"Private RSA key for SSH connection"
            },
            "private_key_passphrase":{
                "type":"string",
                "title":"Private RSA key passphrase"
            }
        }
    },
    "retrieve_keystone_configuration":{
        "type":"object",
        "title": "Keystone configuration",
        "properties":{
            "keystone_config_file": {
                "type": "string",
                "title": "Configuration file path"
            }
        }
    },
    "check_ldap":{
        "type":"object",
        "title": "LDAP settings",
        "properties":{
            "ldap_url": {
                "type": "string",
                "title": "LDAP URL"
```

```
            }
        }
    }
}
```

Parameters:

```
{
    "connect_to_server": {
      "username": "root",
    "password": "password",
    "hostname": "127.0.0.1",
      "port": 22
    },
    "retrieve_keystone_configuration": {
        "keystone_config_file": null
    },
    "check_ldap": {
        "ldap_url": "ldap://"
    }
}
```

Code:

```
import paramiko
import StringIO
from driver import Driver
from oslo_config import cfg

class SSHClient(object):
    def ssh_connect(self, hostname, port, username, password=None, private_key=None,
          private_key_passphrase=None):
        ssh_client = paramiko.SSHClient()
        ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        if private_key:
            private_key = paramiko.RSAKey.from_private_key(private_key, password=private_key_passphrase)
        ssh_client.connect(hostname, port, username=username, password=password, pkey=private_key)
        return ssh_client


class KeystoneLdapControl(Driver,
                  SSHClient):

    keystone_config_file = None
    # Step 1 - Check prerequisites
    def prerequisites(self, inputs):
        self.keystone_config_file = "/etc/keystone/keystone.conf"
        retrieve_keystone_configuration_ti = self.testinstances.get("retrieve_keystone_configuration", None)
        if retrieve_keystone_configuration_ti is not None:
            self.keystone_config_file = retrieve_keystone_configuration_ti.get("keystone_config_path", self.
                keystone_config_file)
```

```
26          check_ldap_ti = self.testinstances.get("check_ldap")
27          assert check_ldap_ti is not None
28          self.ldap_url = check_ldap_ti.get("ldap_url")
29          assert self.ldap_url is not None
30          return True
31
32      # Step 2 - Connects to the remote server
33      def connect_to_server(self, inputs):
34          assert inputs is True
35          ssh_connection_ti = self.testinstances.get("connect_to_server", None)
36
37          assert not ssh_connection_ti is None
38
39          hostname = ssh_connection_ti.get("hostname")
40          port = ssh_connection_ti.get("port")
41          username = ssh_connection_ti.get("username")
42          password = ssh_connection_ti.get("password", None)
43          private_key = ssh_connection_ti.get("private_key", None)
44          if private_key is not None:
45              private_key = StringIO.StringIO(private_key)
46              private_key_passphrase = ssh_connection_ti.get("private_key_passphrase", None)
47
48          assert not password is None or not private_key is None
49
50
51          self.ssh_connection = self.ssh_connect(
52              hostname=hostname,
53              username=username,
54              port=port,
55              password=password,
56              private_key=private_key,
57              private_key_passphrase=private_key_passphrase
58              )
59          return True
60
61      # Retrieves keystone configuration from remote server (OpenStack controller)
62      def retrieve_keystone_configuration(self, inputs):
63          assert self.keystone_config_file is not None
64          _stdin, _stdout, _stderr = self.ssh_connection.exec_command("cat %s" % self.keystone_config_file)
65          lines = _stdout.readlines()
66          return lines
67
68      # Checks the identity driver specified to be ldap, and that the url of ldap
69      # in the [ldap] section is the one specified in the control input
70      # More checks can be added such as ldap query string
71      def check_ldap(self, ldap_config):
72          class MyConfigParser(cfg.ConfigParser):
73              def ___init___(self, filename, ldap_config):
74                  super(cfg.ConfigParser, self).___init___()
75                  self.sections = {}
76                  self._normalized = None
77                  self.section = None
78                  self.ldap_config = ldap_config
79                  self.filename = filename
80
```

```
81                  def parse(self):
82                      return super(cfg.ConfigParser, self).parse(self.ldap_config)
83
84              mcp = MyConfigParser(self.keystone_config_file, ldap_config)
85              mcp.parse()
86              assert mcp.sections.get("identity", None) is not None
87              assert mcp.sections.get("ldap", None) is not None
88              return mcp.sections.get("identity").get("driver") == "ldap" and mcp.sections.get("ldap").get("url") == self.
                    ldap_url
89
90          def close_ssh_connection(self, inputs):
91              try:
92                  self.ssh_connection.close()
93              except:
94                  pass
95              return inputs
96
97
98          def appendAtomics(self):
99              self.appendAtomic(self.prerequisites, lambda: None)
100             self.appendAtomic(self.connect_to_server, self.close_ssh_connection)
101             self.appendAtomic(self.retrieve_keystone_configuration, lambda: None)
102             self.appendAtomic(self.check_ldap, lambda: None)
103             self.appendAtomic(self.close_ssh_connection, lambda: None)
```