A semi-automatic and trustworthy scheme for continuous cloud service certification

Marco Anisetti, Claudio Ardagna *Member, IEEE,* Ernesto Damiani *Senior Member, IEEE,* Filippo Gaudenzi

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.

Index Terms—Assurance, Certification, Cloud, Testing

1 Introduction

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 [1], [2]. The advent of cloud in fact makes traditional techniques inappropriate, because assurance claims and information were 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 [3], *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 [1], [2], [4], and in turn fosters the movement of critical businesses to the cloud. In this paper, 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. 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. Our scheme 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 evaluation 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 allows

• *M. Anisetti, C.A. Ardagna, E. Damiani, F. Gaudenzi are with the Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy. E. Damiani is also with Etisalat British Telecom Innovation Center, Khalifa University of Science, Technology and Research, Abu Dhabi, UAE. E-mail: {firstname.lastname}@unimi.it*

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.

Our contribution is manifold. After describing a certification scheme for the cloud discussing its building blocks and certification process (Sections 3 and 4), we define a matching algorithm supporting dynamic delegation of responsibilities and a semi-automatic certification process, which reduces the burden of certification authorities (Section 5). We then present a continuous and incremental process for the management of a certificate life cycle from issuing to revocation (Section 6). We further implement our algorithm and process in a certification framework, and experimentally evaluate it in terms of quality, performance, utility, and practical usability in a real industrial scenario based on Engineering Ingegneria Informatica S.p.A. ENGpay online payment system, ENGpay in the following (Section 7). Finally, we provide a discussion on a trust model for the cloud emerging from the proposed approach (Section 8).

2 Related Work

Certification schemes aim to provide evidence that a given software system has some non-functional properties and behaves as expected. Their evolution followed the evolution of IT, first focusing on software certification (e.g., [5]), then considering service certification (e.g., [3], [6]), and finally approaching certification in the cloud (e.g. [7], [8]). In the last few years, different approaches to security certification of services have been proposed (e.g., [3], [8]–[11]). Many of them [3], [9] considered static service certification, with no support for continuous certification of evolving applications. Common Criteria scheme [5] was the first approach to incremental certification distinguishing between partial re-evaluation and partial re-certification [11]. However, it provides a manual process and puts a high overhead on developers and certification authorities. Anisetti et al. [10] then provided an approach to continuous and incremental certification of SOA and web services, by extending their test-based security certification scheme in [3].

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 [12] discussed the advantages introduced by a certification approach when integrated in a cloud system, while Chen et al. [13] proposed a cloud security assessment indicator system based on classifying and grading. Lins et al. [1] then presented a conceptual architecture for continuous verification of cloud services discussing its benefits and challenges. Krotsiani et al. [14] proposed an approach based on monitoring for incremental security certification of cloud services. Although their model-based methodology is similar to the one in this paper, it 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. Stephanow et al. [15] 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 and suffers from similar problems as [14]. Stephanow et al. [16] 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 to further reduce the need of human intervention.

The research community has also focused on different solutions supporting trust in the cloud and implementing different trust models for the cloud [7], [17]–[20]. Ryan et al. [17] 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. [18] 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. [19] proposed an approach to deploy and scale cloud services based on security- and performance-related evidence. Wahab et al. [20] proposed a trust-model for cloud services that is completely decentralized. A cloud service trust is built by collecting feedback from its neighbors, which is calculated as the relation between successful interactions over total interactions and spread following a hedonomic coalition game. Differently from the above papers, we describe a trust model based on certification and a running schema for trusted evidence collection.

Another relevant line of research analyzed the problem of cloud testing specifically focusing on Testing-as-a-Service paradigm [21], [22]. Tsai et al. [21] proposed a TaaS design for SaaS combinatorial testing. King et al. [22] proposed a virtual test environment that combines an automated test harness for a cloud service, with the delivery of Test Support-as-a-Service (TSaaS). Gonzales et al. [23] presented a set of tests and attacks, which affect the cloud infrastructure layer and can be used to assess different security properties. In this paper, we address the issues of cloud-based incremental testing in a way similar to the regression testing described by Di Penta et al. [24], where the goal is to re-use, as much as possible, existing test cases. Differently from the approach in this paper, Di Penta et al. [24] focused on identifying new faults, rather than on certifying the non-functional properties of evolving services; moreover, in this paper, incremental testing is driven by certification models and the relative regression is based on them.

To conclude, the approach in our paper develops on the architecture in [25] and the initial trust model in [7] by providing *i)* the methodology for consistency check between

certification model template and instance, *ii)* a complete certification process starting from certificate issuing to complete certificate life cycle management, *iii)* a certification framework implementing the continuous certification process and providing tools for supporting certification authorities, *iv)* a complexity analysis of the proposed approach that evaluates its computational overhead and, in turn, its practical applicability in real industrial scenarios.

3 Certification Building Blocks

We describe the building blocks of our certification scheme for the cloud. Examples in this paper refer to the security domain and consider a cloud-based deployment of online payment system ENGpay on top of OpenStack IaaS.

3.1 Non-Functional Properties

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) [3] or domain-specific vocabularies derived from regulations (e.g., [26]), standards (e.g., [27]), cloud security specifications (e.g., [9]). 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., [3], [9]), we decouple definition of properties from the mechanisms that must be implemented to support them. Non-functional properties are organized in a hierarchy $\mathcal{H}_P=(\mathcal{P},\preceq_P)$ [3], where $\mathcal{P}$ is the set of properties and $\preceq_P$ is a partial order relationship over $\mathcal{P}$. 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 $v_i(a_k)$ of attribute a_k is not specified or $v_i(a_k)\preceq_{a_k} v_j(a_k)$. 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 [3].

3.2 Non-Functional Mechanisms

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 $v_j(a_t)$ is not specified or $v_j(a_t)\preceq_{a_t} v_k(a_t)$. 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 $\{events\}$ 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.

3.3 Target of Certification

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.

3.4 Evidence

A certification process (see Section 4) 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 $ev=\{\{(\theta,Pr,In,EO,Po)\}\}$, where $\{(\theta,Pr,In,EO,Po)\}$ 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) [3] 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

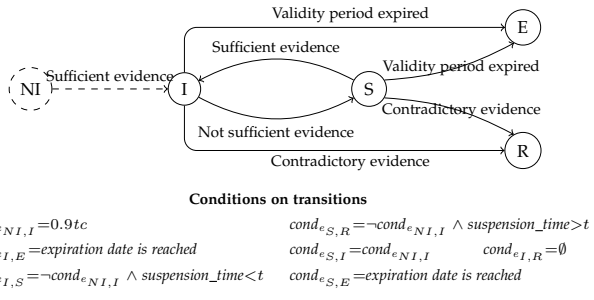

Fig. 1. Life cycle with states Not Issued (NI), Issued (I), Suspended (S), Expired (E), Revoked (R) and examples of conditions on transitions.

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 [3]. Each test case *tc* refers and exercises a given flow ϕ_i.

3.5 Life Cycle

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 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 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.

4 Cloud Certification Process

A certification process for the cloud is a collaborative effort involving *i)* a *service provider* developing cloud-based applications that need to be certified; *ii)* a *cloud provider* that either supports application certification or wants to certify its own cloud services; *iii)* a *certification authority* responsible for the design and definition of certification requirements and methodology; *iv)* an *accredited lab* delegated by the certification authority and responsible for certification activities.[1] We note that the *accredited lab* can be either an online actor using a certification framework to carry out evaluation activities or the framework itself. In the following, when clear from the context, we will call accredited lab both the online actor and the framework.

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 [25] (*post-issuing phase*). It is based on specific machine-readable documents, namely Certification Model (CM) Template, Certification Model (CM) Instance, and Certificate.

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 (see Section 8). 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 4.1 ($\mathcal{T}$). Let us consider the cloud storage service of ENGpay, 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*=($\{\theta_1, \theta_2, \theta_3\}$,<*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

1. An accredited lab is an official emanation of a certification authority carrying out a system evaluation.

$\mathcal{D}_{In}$ 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 concrete implementation of the methodology is a refinement of $\mathcal{T}$, called CM Instance $\mathcal{I}$. 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 $\langle \overline{p}, \overline{ToC}, \overline{m}, \overline{ev}, \overline{l} \rangle$. 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 CM Template signed by the certification authority (see Section 8). 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 5). 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. However, how this information is really used in practice to execute test cases is out of the scope of this paper.

Example 4.2 ($\mathcal{I}$). Let us consider CM Template $\mathcal{T}$ in Example 4.1. 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 $\overline{ToC}$=($\{\theta_1,\theta_2,\theta_3\}$,<*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 4.1 ($\xrightarrow{I}$). 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: $p \xrightarrow{I} \overline{p}$, $ToC \xrightarrow{I} \overline{ToC}$ $m \xrightarrow{I} \overline{m}$, $ev \xrightarrow{I} \overline{ev}$, $l \xrightarrow{I} \overline{l}$.

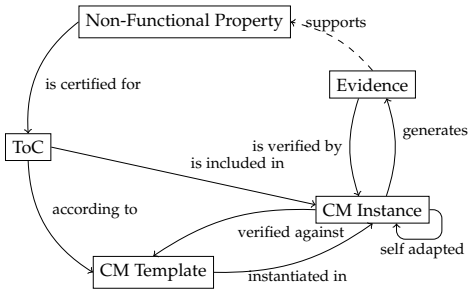

Fig. 2. Conceptual framework

CM Instance $\mathcal{I}$ describes an executable evidence collection process (see Figure 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 $\overline{l}$ (see Section 7.1). 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. A concrete incarnation of Examples 4.1 and 4.2, as well as certicates built on them, are available at http://tinyurl.com/hwyvklb.

Figure 2 shows the conceptual framework at the basis of our certification process, as an extension of the one in [7]. 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 5. 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 2) and therefore award the corresponding certificate.

Based on the conceptual framework, Figure 3 presents our certification process that is composed of five main steps implemented in the certification framework in Section 7.1. 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,

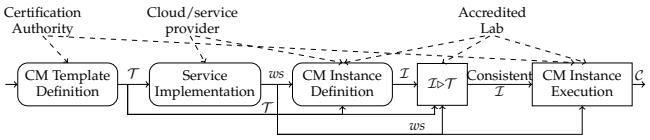

Fig. 3. Certification process: execution steps. Squared boxes represent steps implemented in our framework in Section 7.1, rounded boxes steps for which we provide tools or guidelines for cloud providers, accredited labs, or CA.

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.

5 Consistency Check

The consistency check $\mathcal{I} \triangleright \mathcal{T}$ is the main pillar of the certification process in Figure 3. 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. 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 $\mathcal{I}=\langle \overline{p}, \overline{ToC}, \overline{m}, \overline{ev}, \overline{l} \rangle$ 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.

5.1 Non-functional 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**: $\overline{p}$ 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 3.1). Property verification checks that the property to be certified $\overline{p}$ is equal to/stronger than p.
- **ToC verification**: $\overline{ToC}=(\overline{\Theta}, \overline{b})$ 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 3.3), *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 5.1. 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 ENGpay system 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.

5.2 Evidence collection model verification

We aim to verify the consistency between the STS-based models $m \in \mathcal{T}$ and $\overline{m} \in \mathcal{I}$. Our verification is successful, meaning that $\overline{m}$ is a valid instance of m, if the linear independent paths modeling the testing flows $\Phi(\overline{m})$ 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 $G^{\overline{m}}(V^{\overline{m}}, E^{\overline{m}})$ of models m and $\overline{m}$, 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 $\overline{m}$ as a single mechanism. This is due to the fact that $\overline{m}$ specifies a variety of functional-related mechanisms that cannot be known a priori by m.

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

Definition 5.1 ($m \xrightarrow{I} \overline{m}$). Let $G^m(V^m, E^m)$ be the quotient graph of m, $G^{\overline{m}}(V^{\overline{m}}, E^{\overline{m}})$ be the quotient graph of $\overline{m}$. Also, let θ_i be the mechanism associated with each $v_i \in V^m \cup V^{\overline{m}}$. $\overline{m}$ is a valid instance of m, denoted $m \xrightarrow{I} \overline{m}$, iff there exists an isomorphism $f: V^m \rightarrow V^{\overline{m}}$, such that the following conditions hold:

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

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

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 5.2. 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 ENGpay system having a model $\overline{m}$ for secure data exchange as a sequence of a functional mechanism for the selection and configuration of the ENGpay 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 $\overline{m}$ keeps the first two mechanisms as they are and merges the two functional mechanisms at the end of $\overline{m}$ in a single mechanism. $m \xrightarrow{I} \overline{m}$ because G^m and $G^{\overline{m}}$ 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 $G^{\overline{m}}$ according to Example 5.1.

5.3 Evidence verification

Evidence verification 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 $\{ev_1,...ev_n\}$ of possible evidence. Evidence ev and $\overline{ev}$ are composed of a set of test cases $tc=\{(\theta,Pr,In,EO,Po)\}$ (see Section 3.4). 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 $\overline{m}$ 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 $\overline{m}$ in a single mechanism, and *ii)* each quintuple (θ,Pr,In,EO,Po) in $tc \in ev$, with $\theta=(Functional,\{\})$, 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 5.2 (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 5.2 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 5.3. Starting from Example 5.2, 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 ENGpay, 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 5.2.

Following Definition 5.2, evidence $\overline{ev} \in \mathcal{I}$ is a valid instance of evidence $ev \in \mathcal{T}$ as follows.

Definition 5.3 ($ev \xrightarrow{I} \overline{ev}$). Let $ev=\{tc_1,...,tc_k\}$ be an evidence of $\mathcal{T}$ and $\overline{ev}=\{\overline{tc}_1,...,\overline{tc}_n\}$ 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 $\overline{m}$ and corresponding path ϕ_j of m according to Definition 5.1, $\forall tc_r \in ev$ associated with ϕ_j, $\exists tc_t \in \overline{ev}$ associated with $\overline{\phi}_i$ s.t. $tc_r \times tc_t$.

Definition 5.3 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 5.2.

5.4 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 $\overline{l}$ in $\mathcal{I}$ must then adhere to the abstract lifecycle l in $\mathcal{T}$. $\overline{l}$ is a valid instance of l, denoted $l \xrightarrow{I} \overline{l}$, *iff* $G^{\overline{l}}(V^{\overline{l}},E^{\overline{l}})$ is isomorphic to $G^l(V^l,E^l)$ and each condition labeling edges in $\overline{l}$ is more restrictive of the corresponding in $\overline{l}$, as formalized by the following definition.

Definition 5.4 ($l \xrightarrow{I} \overline{l}$). Let $G^l(V^l,E^l)$ be the lifecycle in $\mathcal{T}$ and $G^{\overline{l}}(V^{\overline{l}},E^{\overline{l}})$ be the lifecycle in $\mathcal{I}$. $G^{\overline{l}}(V^{\overline{l}},E^{\overline{l}})$ is a valid instance of $G^l(V^l,E^l)$, denoted $G^l(V^l,E^l) \xrightarrow{I} G^{\overline{l}}(V^{\overline{l}},E^{\overline{l}})$, *iff* there exists an isomorphism $f:V^l \rightarrow V^{\overline{l}}$, such that:

1) $\forall v_i \in V^l$, $\exists f(v_i) \in V^{\overline{l}} \wedge label(v_i)=label(f(v_i))$;

2) $\forall(v_i,v_j)\in E^l$, $(f(v_i),f(v_j))\in E^{\bar{l}}$ and is such that $cond_{(f(v_i),f(v_j))}$ is more restrictive than $cond_{(v_i,v_j)}$.

Condition 1 states that each vertex v_i in G^l, representing a state of a certificate, should have a corresponding vertex $f(v_i)$ 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 1). Existing approaches for the comparison of boolean expressions (e.g., [28]) can be integrated within lifecycle verification.

Consistency check $\mathcal{I}\triangleright\mathcal{T}$ can then be modeled 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 5.5 ($f^{\triangleright}$). 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*), $m\xrightarrow{I}\overline{m}$ (*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 4.1 and 4.2 can be found at http://tinyurl.com/hwyvklb. 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 this paper. In Section 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.

6 Certificate Life Cycle Management

Certificate life cycle management relies on evidence collection and CM Instance validity check to verify the validity of a CM Instance w.r.t. the corresponding CM Template for certificate issuing (Section 6.1) and to monitor changes to either CM Templates or CM Instances for certificate adaptation (Section 6.2) as discussed in the following.

6.1 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 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.

6.2 Certificate Adaptation

Certificate adaptation is built around states I, S, R, and E of $\bar{l}$ in Figure 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 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, 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 5.

Table 1 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*).

6.2.1 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 incremental 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.

TABLE 1
Adaptation summary

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

- *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 $\bar{l}'$ is defined in $\mathcal{I}'$, no operations are needed *iff* $\bar{l}'\xrightarrow{I}\bar{l}$. 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 $\bar{l}'$ is defined in $\mathcal{I}'$, no operations are needed *iff* $\bar{l}'\xrightarrow{I}\bar{l}$.

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.

6.2.2 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 (US-Cert – https://www.us-cert.gov/) identifies a new vulnerability for a given *ToC*, which calls for $\mathcal{T}$ 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.

7 Experimental Evaluation

We give a brief overview of our certification framework (Section 7.1), present a complete experimental evaluation of our approach in terms of efficiency and quality (Section 7.2), and discuss the utility and practical usability of our approach for life cycle management in a real industrial scenario (Section 7.3), where ENGpay online payment system is checked for compliance against PCI-DSS standard. 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, installing the whole certification framework. 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 certification framework, are available at http://tinyurl.com/hwyvklb.

7.1 Certification Framework

Our certification framework extends the certification framework in [29] to implement the process in Figure 3, providing

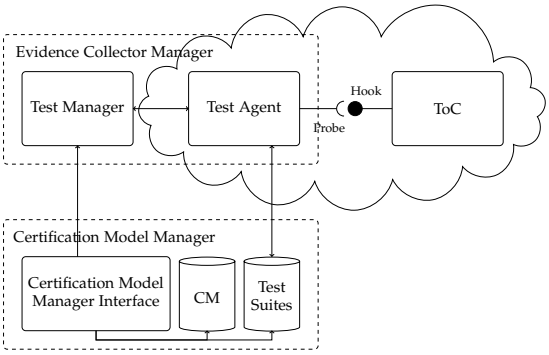

Fig. 4. A simplified view of the certification framework architecture

a set of tools and guidelines for designing CM Templates, generating CM Instances, consistency check, and certification management. It is composed of two main components: *i) Certification Model Manager* that deals with CM generation, selection, verification, *ii) Evidence Collector Manager* that parses the CM Instance $\mathcal{I}$ and drives the testing process by injecting test cases of $\overline{ev}$ in $\overline{ToC}$.

Certification Model Manager is the main interface for certification authorities and accredited labs managing a certification process. It implements the repository of certification models and building blocks in Sections 3 and 4, such as the hierarchies of properties $\mathcal{H}_P$ and mechanisms $\mathcal{H}_M$. It also implements the *Test Suites Repository*, where all test suites are stored with corresponding evidence and have been kept up to date. It further offers internal functionalities (e.g., consistency check – $f^{\triangleright}$, certificate adaptation) supporting the *Evidence Collector Manager* in the management of the certification process and certificate life cycle (step $\mathcal{I}\triangleright\mathcal{T}$ in Figure 3).

Evidence Collector Manager implements a master-slave architecture for evidence collection driven by CM Instance $\mathcal{I}$ [29] (step *CM Instance Execution* in Figure 3). It is composed of a *Test Manager* (TM), the master owning the collection process, and one or more *Test Agents* (TAs), the slaves responsible for testing specific flows $\Phi(\overline{m})$ over $\overline{ToC}$. TM *i)* provides interfaces to manage (e.g., start, stop) the collection process, *ii)* configures the evidence collection process and initializes TAs according to configurations, models, and test cases in $\mathcal{I}$, and *iii)* manages the certificate life cycle according to $\overline{l}$. TA retrieves the relevant test cases in $\overline{ev}$ from the *Test Suites Repository*, executes them on the ToC, and returns the results of their execution to TM.

Figure 4 shows the data flow between *Certification Model Manager* and *Evidence Collection Manager*. *Certification Model Manager* is externally accessed through a restful interface, which permits to manage CM Templates and CM Instances, and communicates with TM and TA(s) in the *Evidence Collector Manager*. A certification process starts when the *Certification Model Manager* sends a valid CM Instance $\mathcal{I}$ to TM. Upon receiving $\mathcal{I}$, TM parses it and forwards each of its elements, except property $\overline{p}$, to TA(s). TA(s) retrieves the test cases in $\overline{ev}$ of $\mathcal{I}$ from *Test Suites Repository* and manages the testing activities. The testing activities are executed by one or more probes that access the ToC through a hook. TA(s) sends collected evidence back to TM when available, which aggregates it and eventually triggers a life cycle transition.

7.2 Consistency Check Algorithms

We implemented the *exhaustive algorithm* of our consistency check function $f^{\triangleright}$ as 5 consecutive verification steps according to Definition 5.5. We called it exhaustive because the evidence collection model verification is carried out by exhaustively searching if, among all possible permutations of flows $\Phi(\overline{m})$ in $\mathcal{I}$, there exist one or more isomorphisms (Definition 5.1) 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[2] 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.

Heuristic 1: k-matching. Evidence collection model verification is carried out flow by flow and aims to find multiple matching, isomorphisms in Definition 5.1, between m of $\mathcal{T}$ and $\overline{m}$ of $\mathcal{I}$. It logically traverses the permutation tree of the flows $\phi_j{\in}\Phi(\overline{m})$ 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* $\phi_j{\in}\Phi(\overline{m})$ matches the corresponding $\phi_d{\in}\Phi(m)$ according to Definition 5.1. The resulting sub-tree includes zero or more isomorphisms between m and $\overline{m}$, represented as paths of length β. In the worst case scenario, the algorithm has an exponential asymptotic behavior $O(k^{\beta-k+1}{\cdot}(k{-}1)!)$, 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 $\Phi(\overline{m})$, 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*.

7.2.1 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 in Section 6. We automatically generated, using the tool available at http://tinyurl.com/jxoubsb, 19 CM Templates $\langle p,ToC,m,ev,l\rangle$ (see Section 4), varying the number β of flows between 1 and 19

2. Heuristics pseudocode is available at http://tinyurl.com/zkqs9vn.

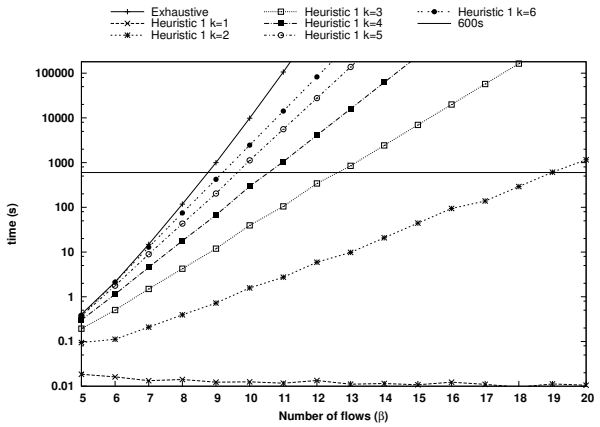

Fig. 5. Execution time (log scale) varying the number of flows

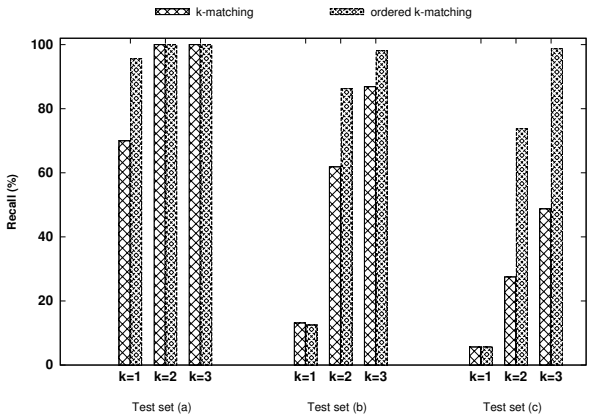

Fig. 6. Quality evaluation considering three instantiations of our k-matching algorithm with k=1, k=2, and k=3.

(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 at http://tinyurl.com/jxoubsb. 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.

Figure 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 7.2. 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.

7.2.2 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 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 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.

7.3 Life Cycle Management Evaluation

All cloud-based systems with strict non-functional requirements can benefit from a trustworthy certification approach. These systems include cloud-based systems managing sensitive data and critical application processes, and cloud-based systems that need to show verifiable compliance with standards or regulations. We therefore discuss the utility and practical usability of our approach in a real industrial scenario based on the ENGPay e-payment system deployed in the cloud. In particular, we show how our certification framework has been instantiated and used for security compliance verification of ENGPay against the Payment Card Industry Data Security Standard (PCI DSS). Certification-based compliance verification was first envisioned by Accorsi et al. [30]. Here, we briefly describe the collaborative effort for compliance verification between us, acting as certification authority/accredited lab using the certification framework, and Engineering Ingegneria Informatica S.p.A. ENGPay team, acting as a service provider. PCI DSS is a global standard established by financial organizations to protect cardholder data and information linked to users' personal data. It is based on a set of requirements (details are available at https://it.pcisecuritystandards.org/security_standards/) calling for online and continuous evaluation activities, to verify storage integrity, confidentiality of data in transit, and the like.

We considered a specific cloud-based deployment of ENGpay system designed for PCI DSS compliance verification and testing. It was deployed in three virtual machines, one for each ENGPay service, on top of OpenStack, as follows: *i) ENGpay HUB service*, providing the centralized functionalities to manage users' needs (e.g., start a transaction), *ii) Acquirer-Issuer service*, providing functionalities for the Cleaning&Settlement process, *iii) Selfcare Data management*, providing functionalities to manage card holder data and all transaction information. For conciseness, we focused on PCI DSS requirement 4 *Encrypt transmission of cardholder data*, which requires confidentiality of data in transit. We defined a CM Template $\mathcal{T}$ for assessing PCI DSS requirement 4 asking for secure communications based on HTTPS both at infrastructure and application layers, and instantiated it in CM Instance $\mathcal{I}$ for compliance verification of ENGpay. Our approach to life cycle management was then used to continuously verify the compliance of ENGPay against PCI DSS and trigger adaptation whenever necessary. In the following, we assume that a certificate was issued for ENGPay system according to the CM Instance for PCI DSS requirement 4, and discuss three adaptation scenarios.

Adapted template $\mathcal{T}'$ of $\mathcal{T}$. A new bug is discovered on the HTTPS implementation used by ENGpay (e.g., Heartbleed Bug of openSSL). This new event triggers an adaptation of $\mathcal{T}$, which results in a new template $\mathcal{T}'$ including additional test cases for evaluating the robustness of the application against the bug. The change in $\mathcal{T}$ triggers a change in $\mathcal{I}$ and the execution of the new test cases on the *ToC* (i.e., ENGPay).

Adapted instance $\mathcal{I}'$ of $\mathcal{I}$: Service configuration event. ENGPay system changes the HTTPS mechanism used by *Acquirer-Issuer service*, deploying a new version of openSSL. This change triggers an adaptation of CM Instance $\mathcal{I}$, which results in a new CM instance $\mathcal{I}'$ including the details and test cases for evaluating the new mechanism. Upon verifying that $\mathcal{I}'$ is still consistent with $\mathcal{T}'$ ($\mathcal{I}' \triangleright \mathcal{T}'$), the incremental certification process tests the HTTPS mechanism of Acquirer-Issuer service only.

Adapted instance $\mathcal{I}''$ of $\mathcal{I}'$: Cloud management event. Finally, ENGPay components are migrated to a new OpenStack infrastructure. CM instance $\mathcal{I}'$ is further modified and distributed as a new instance $\mathcal{I}''$. After verifying that $\mathcal{I}''$ is still consistent with $\mathcal{T}'$ ($\mathcal{I}'' \triangleright \mathcal{T}'$), the incremental certification process tests the OpenStack infrastructure. We note that, in case our application is moved to an OpenStack that does not correctly implement SSL for internal communications, the certificate is revoked. We also note that if the new OpenStack is such that $\mathcal{I}'' \not\triangleright \mathcal{T}'$, a new consistent template is searched and, if found, partial re-certification executed.

We showed how our certification-based approach can be used to provide verifiable evidence of a service behavior at the basis of compliance verification against existing standards, and how the life cycle management can be used for handling runtime cloud events. To show the generality and interoperability of our approach, we re-used test cases applied in a different industrial scenario in [25]. This choice shows that requirements specified in a CM Template can be cross services and instantiated in different CM Instances. We did not report on performance improvement of our certificate life cycle management with respect to traditional certifications, because performance is highly dependent on the proposed scenario and can be affected by our own choices, limiting the impartiality of the results. Interested readers can see examples of real performance results in [25].

8 Discussion on Trust Model

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.

8.1 Chain of Trust

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 [5] certified product, an assurance level value in 1-7).

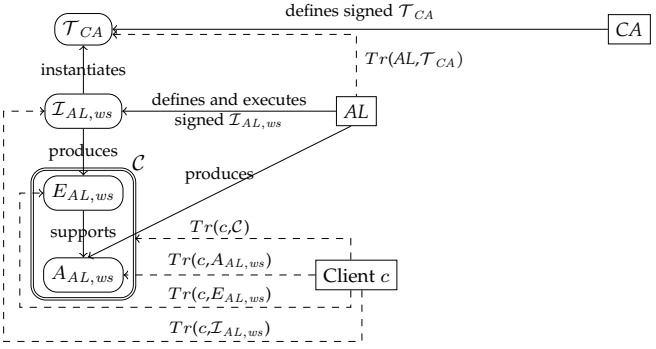

Fig. 7. Chain of Trust for cloud certification

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 4, 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 7 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., $Tr(c,E_{AL,ws})$).

8.2 Chain of Trust and Life Cycle Management

Our chain of trust supports the certificate life cycle in Figure 1, and both issuing and post-issuing phases in Section 4.

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 6.2, 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 [10], 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.

9 Conclusions

We 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. We finally presented the implementation of our approach in a certification framework and evaluated its performance, quality, and utility in a real industrial scenario.

10 Acknowledgments

We would like to thank Domenico Presenza (Engineering Ingegneria Informatica S.p.A.) for its continuous support during the compliance verification of ENGPay system.

References

[1] 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.
[2] I. Windhorst and A. Sunyaev, "Dynamic certification of cloud services," in *Proc. of ARES 2013*, Regensburg, Germany, September 2013.
[3] 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.
[4] Microsoft, *Trusted Cloud*, https://www.microsoft.com/en-us/server-cloud/trusted-cloud/overview.aspx, Accessed June 2016.
[5] D. Herrmann, *Using the Common Criteria for IT security evaluation*. Auerbach Publications, 2002.
[6] 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.
[7] 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.
[8] M. Krotsiani, G. Spanoudakis, and K. Mahbub, "Incremental certification of cloud services," in *Proc. of SECURWARE 2013*, Barcelona, Spain, August 2013.
[9] *CSA Security, Trust & Assurance Registry (STAR)*, Cloud Security Alliance (CSA), https://cloudsecurityalliance.org/star/, Accessed June 2016.
[10] 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.
[11] C. Criteria, *CCRA Supporting Document 2004-02-009 Assurance Continuity*, February 2004, http://www.commoncriteriaportal.org/files/supplements/2004-02-009.pdf.
[12] A. Sunyaev and S. Schneider, "Cloud services certification," *Communications of the ACM*, vol. 56, no. 2, pp. 33–36, February 2013.
[13] 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.
[14] M. Krotsiani, G. Spanoudakis, and C. Kloukinas, "Monitoring-based certification of cloud service security," in *Proc. of C&TC 2015*, Rhodes, Greece, October 2015.
[15] 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.
[16] 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.
[17] 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.
[18] Q. Malluhi and K. M. Khan, "Establishing trust in cloud computing," *IT Professional*, vol. 12, pp. 20–27, 2010.
[19] 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.
[20] 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.
[21] W. T. Tsai, G. Qi, L. Yu, and J. Gao, "TaaS (testing-as-a-service) design for combinatorial testing," in *Proc. of SERE 2014*, San Francisco, CA, USA, June–July 2014.
[22] T. M. King and A. S. Ganti, "Migrating autonomic self-testing to the cloud," in *Proc. of IEEE ICSTW 2010*, Paris, France, April 2010.
[23] D. Gonzales, J. Kaplan, E. Saltzman, Z. Winkelman, and D. Woods, "Cloud-trust - a security assessment model for infrastructure as a service (IaaS) clouds," *IEEE TCC*, vol. PP, no. 99, pp. 1–1, 2015.
[24] M. Di Penta, M. Bruno, G. Esposito, V. Mazza, and G. Canfora, "Web services regression testing," in *Test and Analysis of web Services*. Springer, 2007, pp. 205–234.
[25] 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.
[26] *Health Insurance Portability and Accountability Act (HIPAA)*, U.S. Department of Health & Human Services, November 2015, http://www.hhs.gov/ocr/privacy/hipaa/understanding/.
[27] *ISO/IEC 27001 - Information security management*, ISO/IEC, November 2015, http://www.iso.org/iso/home/standards/management-standards/iso27001.htm.
[28] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE TC*, vol. 100, no. 8, pp. 677–691, 1986.
[29] M. Anisetti, C. Ardagna, E. Damiani, and F. Gaudenzi, "A certification framework for cloud-based services," in *Proc. of SAC 2016*, Pisa, Italy, April 2016.
[30] R. Accorsi, D.-I. L. Lowis, and Y. Sato, "Automated certification for compliant cloud-based business processes," *Business & Information Systems Engineering*, vol. 3, no. 3, pp. 145–154, 2011.

Marco Anisetti Marco Anisetti is an Assistant Professor at the Università degli Studi di Milano, Italy. His research interests are in the area of Computational Intelligence, and its application to the design of complex systems and services. Recently, he has been investigating the adoption of Computational Intelligence techniques in the area of security mechanisms for distributed systems, and software/service testing/monitoring for certification.

Claudio A. Ardagna is an Associate Professor at the Università degli Studi di Milano, Italy. His research interests are in the area of big data analytics, cloud security and assurance, and cloud performance. He is the recipient of the ERCIM STM WG 2009 Award for the Best PhD Thesis on Security and Trust Management. He has co-authored the Springer book "Open Source Systems Security Certification". The URL for his web page is http://www.di.unimi.it/ardagna

Ernesto Damiani is a Full Professor at the Università degli Studi di Milano, where he leads the SESAR research lab, and the leader of the Big Data Initiative at the EBTIC/Khalifa University in Abu Dhabi, UAE. He is the Principal Investigator of the H2020 TOREADOR project. He was a recipient of the Chester-Sall Award from the IEEE IES Society (2007). He was named ACM Distinguished Scientist (2008) and received the Stephen S. Yau Services Computing Award (2016).

Filippo Gaudenzi received his master degree in Informatics Engineering (2012) at the Università Politecnica di Ancona. Currently, he is a Ph.D. Student at the Università degli Studi di Milano, where he worked as a Assistant Researcher within the FP7 project CUMULUS. He worked as R&D Engineering at INRIA - Bretagne Atlantique within the FP7 Contrail project. His main research interests are in the areas of cloud, security, and monitoring and testing tools.