A Certification Technique for Cloud Security Adaptation

Claudio A. Ardagna
Università degli Studi di Milano
Crema, Italy
claudio.ardagna@unimi.it

Rasool Asal, Ernesto Damiani
EBTIC – Khalifa University
Abu Dhabi, UAE
rasool.asal@bt.com
ernesto.damiani@kustar.ac.ae

Nabil El Ioini, Claus Pahl
Free University of Bozen
Bolzano, Italy
{nelioini,claus.pahl}@unibz.it

Theo Dimitrakos
BT and the University of Kent
UK
theo.dimitrakos@bt.com

***Abstract*—Unpredictability of cloud computing due to segregation of visibility and control between applications, data owners, and cloud providers increases tenants' uncertainty when using cloud services. Adaptation techniques become fundamental to provide a reliable cloud-based infrastructure with definite behavior, which preserves a stable quality of service for tenants. Existing adaptation techniques mostly focus on performance properties and are based on unverifiable evidence, which is collected in an untrusted way. In this paper, we propose a security-oriented adaptation technique for the cloud, based on evidence collected by means of a reliable certification process. Our approach adapts the cloud to maintain stable security properties over time, by continuously verifying certificate validity. It uses the output of verification activities to index a feature model, where equivalent configurations are used as the basis for adaptation. We also provide an analysis of the approach on British Telecommunications (BT) premises.**

***Keywords*-Adaptive cloud, Certification, Feature models**

I. Introduction

The cloud computing paradigm is subverting the traditional IT composed of static and on-premise resources and applications, part of a physical infrastructure owned by a single enterprise. It provides a dynamic and evolving environment where applications, platforms, and infrastructures are released as a service on a pay-as-you-go basis to remote tenants [1]. Cloud comes with several advantages in terms of flexibility, ease-of-use, and reduced costs, while its adoption is often impaired by the difficulties for customers to understand the behavior of a service/application in such a dynamic environment. These difficulties are partly due to segregation of visibility and control between applications, data owners, and cloud providers, which may introduce a level of uncertainty and limit predictability of the overall IT system behavior, when compared to fully managed IT systems. Cloud services are then affected by continuous context changes and (new and unexpected) cloud events, potentially distancing the observed cloud service behavior by the expected one at runtime.

To fully unleash the potential of the cloud, cloud service management needs to be automated at all stack layers and integrated with intelligent techniques for dynamic provisioning. The running system may in fact need to adapt some of its configurations to handle changes that can affect the underlying environment or user requirements. Adaptation techniques and autonomic models have then been proposed to build a reliable cloud with consistent behavior [2], which reacts to context changes and events to preserve a stable quality of service for tenants. Current approaches [3], [4] mainly focus on adapting the cloud according to performance and availability properties, by extending resource management techniques with scalability, elasticity, and reliability algorithms. They are often provided as a black box, where events triggering adaptation activities, as well as monitoring results, are often not available to end users. The end users are expected to trust information becoming available at deployment or access time, with reduced visibility on the reasons triggering an adaptation. While this may not represent a problem for normal users, it represents a fundamental issue for users in critical security domains and often prevents their movement to the cloud.

The approach in this paper addresses the above problems by specifying an adaptation technique driven by trustworthy and verifiable evidence. Evidence is collected using the certification process in [5], which aims to shed light on cloud behavior, increasing the transparency of cloud backend working. Our approach preserves stable security properties for a cloud system, by tuning the cloud configurations to guarantee the validity of the certification process and corresponding certificates over time. System modifications affecting the certification process and certified security properties are precisely pinpointed and used to query a feature model specifying equivalent cloud configurations.

The paper contribution is in the definition of a security adaptation technique based on a certification process (Sections III and IV) and feature model (Section V). The proposed technique builds on a Monitor-Analyze-Plan-Execute (MAPE) control loop process and has the threefold advantage of *i)* providing a user-informed adaptation, *ii)* supporting an automatic adaptation enforcement (Section VI), *iii)* maintaining a stable quality of service based on trustworthy evidence coming from certification activities. A first evaluation of the proposed approach has been conducted on British Telecommunications (BT) premises.

Table I
AN EXAMPLE OF SECURITY PROPERTIES

Property $\hat{pr}$	**Class**	**Level**				
		l_1	l_2	l_3	l_4	l_5
Data access level	C	Access only from within the system (Rule 1)	Access to data from authorized personnel only (Rule 2)	Access to data from authorized personnel for regular operations and administrators for non-regular operations (Rule 3)	Access to data from administrators for exceptional operations only (Rule 4)	Access to data from owners only (Rule 5)
Data exchange confidentiality	C	Encryption with TLS1.0	Encryption with TLS2.0	Encryption with TLS3.0	–	–
Data storage confidentiality	C	Weak encryption or service isolation	Medium encryption or VM isolation	Strong encryption or hardware isolation	–	–
Data alteration detection	I	Auditing	Any access is mediated by an access control system	All data are signed	–	–
Percentage of uptime	A	50≤Uptime<95	95≤Uptime<99	Uptime≥99	–	–

II. BASIC CONCEPTS AND REFERENCE SCENARIO

This section presents our basic concepts and reference scenario.

A. MAPE Control Loop

The components of an autonomic system need to be aware of changes to their state and environment, and appropriately react to them, with little or no involvement of the users. At the core of an autonomic system is a 4-phase control loop called MAPE. MAPE derives from the four phases composing its process, namely Monitor-Analyze-Plan-Execute, which are continuously and sequentially executed in a loop [6]. The phase *Monitor* keeps constantly checking the status of the system and reports the collected data. The phase *Analyze* is the brain of the control loop, where collected data are analyzed. The goal of this phase is to find anomalies or inconsistencies in the collected data by performing different types of data analysis depending on the system context. The phase *Plan* builds a plan of adaptation based on the outputs of phase analysis. It is crucial to select the optimal system adaptation. Once the plan is ready, the phase *Execute* performs the planned actions.

B. Security Properties

A security property pr is a pair ($\hat{pr}$,l), where $\hat{pr}$ is the property name and l is the property level modeling the strength of the supported property. Level l refines the property in terms of general objectives to be supported by the system under consideration, which are independent from the specific security mechanisms. In this paper, we consider properties belonging to the classes confidentiality, integrity, and availability (CIA) [7]. Table I presents for each class examples of properties used as a driver for adaptation techniques.

Security properties represent the target of our adaptation system. Each level is used to define configuration constraints on security mechanisms supporting the considered property. Clearly, the same property level can be achieved by means of different sets of security mechanisms. For instance, each of the three levels of property data storage confidentiality can be supported by either deploying an encryption algorithm or by providing tenant isolation.

C. Reference Scenario

Our reference scenario is an e-health SaaS (EHS) system deployed on top of a public cloud. It is composed of three services distributed over the cloud, each responsible for a specific EHS functionality: *EHS-Patients* responsible for patient management, *EHS-Pharmacy* responsible for medicine and pharmacy management, *EHS-Users* responsible for doctors and nurses management. The reference scenario involves different actors with a different level of control and visibility of the cloud: a cloud provider, managing the overall cloud infrastructure, and three service providers, each managing one of the EHS components and related data. Five main security requirements have been defined for EHS components: [R1] all data exchanged between the different EHS components must be encrypted (property data exchange confidentiality), [R2] all data stored by EHS components must be encrypted (property data storage confidentiality), [R3] an access control system must mediate each request to components EHS-Patients and EHS-Pharmacy providing identification, authentication, and authorization functionalities (property data access level), [R4] all data stored by components EHS-Patients and EHS-Pharmacy must be always signed (property data alteration detection), [R5] the system must be available at least 99% of the time (property percentage of uptime). We note that requirement annotation imposes the definition of constraints on single internal mechanisms implementing the external components (Section IV).

In the following sections, we present how MAPE is implemented in our adaptation approach based on the architecture in [8] and integrates a security certification process.

III. CERTIFICATION PROCESS

The phase monitor of the MAPE control loop consists of the cloud security certification process in [5]. The certification process is managed by a *certification authority* with the support of an *accredited lab* responsible for all evaluation activities. It receives as input the *ToC*, representing the system under verification, the security property pr to be certified, and the list of evaluation activities to be executed. All this information is specified in a machine-readable *certification model*, which is executed on a specific *ToC*

to certify a property pr. After executing the certification model, the process returns as output a certificate describing a set of evidence e proving the support of pr by *ToC*. Then, upon certificate issuing, additional evidence is continuously collected to verify the validity of the certification process and corresponding certificate in production, and in turn the consistency between the observed and expected *ToC* behavior at runtime. Evidence is collected by probes running in the cloud and having access to the *ToC*.

In the following, for clarity and without loss of generality, we consider a simplified certification model, which consists of the system model driving the collection of evidence e at the basis of a certification process. The model refers to pr and *ToC*, and represents the execution paths of the *ToC* as the concatenation of security and functional mechanisms deployed at different layers of the cloud stack. We argue that, as advised by cloud software testing practitioners [9], the loss of generality due to this simplification is limited as no deployed mechanisms of the real system are omitted. Similarly to testing models [9], a certification model can be represented as a direct acyclic graph, where each vertex refers to a mechanism (e.g., access control, encryption, functional mechanisms) and is annotated with a set $\{c_1,\ldots,c_n\}\in\mathcal{C}$ of constraints on cloud configurations, and each edge is annotated with a function call f_i. A cloud configuration refers to a precise setup of a cloud environment and drives cloud service behavior. Formally, a certification model is a system model defined as follows.

Definition 1 (Certification Model $\mathcal{M}$): A certification model is a direct acyclic graph $G^{\mathcal{M}}(V,E,\lambda)$, where a vertex $v_i\in V$ refers to a mechanism, an edge $(v_i,v_j)\in E$ is annotated with function call f_i to the mechanism represented by v_j, and $\lambda:V\rightarrow\mathcal{C}$ is a labeling function that associates a set $\{c_1,\ldots,c_n\}\in\mathcal{C}$ of cloud configuration constraints with each $v_i\in V$.

We note that each function call annotating an edge in $G^{\mathcal{M}}$ triggers a state transition and corresponding mechanism execution. We also note that, to support our adaptation technique in Section V, λ assigns $\lambda(v)$ to each vertex v, corresponding to a set $\{c_1,\ldots,c_n\}$ of alternative cloud configuration constraints c_i annotated on the mechanism represented by v. At least one constraint c_i must be satisfied by mechanism at vertex v to behave correctly and support property pr. A single c_i can be defined as follows.

Definition 2 (Cloud configuration constraint): A cloud configuration constraint c_i is a conjunctive boolean formula of expressions of the form *op*(*attr*,*value*), where *op* is an operator in $\{=,\neq,<,>,\leq,\geq,\in\}$, *attr* represents a configuration attribute referring to a security mechanism, and *value* a (set of) value for the given attribute.

Configuration constraints derive from properties in Table I and are distributed over the model by the certification authority. As an example, in case property data storage confidentiality is selected at level 3 (meaning that stored data must be encrypted or data storage must be kept isolated), a possible annotation for data storage is $=$(*isolation*,hardware isolation) or, as an alternative, the system model must specify an encryption mechanism protecting the data storage and annotated with constraint $\geq$(*encryption_algorithm*,"AES256-SHA"), that is, AES256-SHA or stronger algorithms must be supported.

IV. Certification Process Analysis

The phase analyze of the MAPE control loop is a 2-step phase. The first step (certification model consistency) verifies the consistency between the system behavior and the system model. The second step (constraint verification) analyzes the need of adaptation.

A. Step 1: Certification Model Consistency

Step 1 receives as input the certification model (Definition 1) and data on system behavior collected by probes in phase monitoring (Section III). The goal of this step is to verify the correct behavior of the system by matching the collected evidence against the certification model. Collected evidence is represented as execution traces over paths in the certification model. A path is formally defined as follows.

Definition 3 (Path p_i): Given a certification model, a path p_i is a sequence of vertices $\langle v_0,\ldots,v_n\rangle$, s.t. $\forall_{j=0}^{n-1} v_j\in V$, $\exists$ an edge $(v_j\times f_j\rightarrow v_{j+1})\in E$.

We note that f_j represents the function call assigned to each $(v_i,v_j)\in E$. A trace that is matched against a path is formally defined as follows.

Definition 4 (Trace t_i): An execution trace $t_i\in\mathcal{T}$ is a sequence $\langle f_1,\ldots,f_n\rangle$ of actions, where f_j is a service/operation execution.

A verification function *MV* is defined and takes as input the system model and the collected evidence e, and produces as output either *success* (1), if the evidence conforms to the system model, or *failure* (0), otherwise, with a description of the type of inconsistency found. We note that, while *MV* could not verify all system behaviors, it always computes a verification result according to the collected evidence. *MV* is based on *consistency relation* $\equiv$ between collected evidence and certification model, defined as follows.

Definition 5 (Consistency Relation $\equiv$): Given a trace $t_i=\langle f_1,\ldots,f_n\rangle\in\mathcal{T}$ and $p_j=\langle v_0,\ldots,v_n\rangle\in\mathcal{M}$, $t_i\equiv p_j$ iff $\forall f_k\in t_i$, $\exists$ an edge $(v_{k-1},v_k)\in E$ annotated with action f_p s.t. f_k and f_p refer to the same service/operation.

The results of the consistency relation are given as input to the second step of this phase, which evaluates the need to adapt configuration parameters on the basis of certification model in Definition 1 and/or constraints in Definition 2.

B. Step 2: Constraint Verification

A failure returned by the consistency relation means that there is an inconsistency between the certification model and

the execution traces. This scenario has been tackled in our previous work in [10] and requires a system adaptation.

Some additional verification activities are executed by probes to verify the configuration constraints c_i specified on the vertices of each path p_j. Configuration constraint verification is a function CV that takes as input a path p_j and produces as output either *success* (1), if each configuration constraint in the corresponding path is satisfied, or *failure* (0), otherwise, with the list of vertices violating configuration constraints. Constraints on each vertex are evaluated by probes using the appropriate mechanism (e.g., check configuration files, monitor configuration-dependent execution traces). If the configuration constraints do not match the in-production system configurations, the corresponding vertices are added to a list of misconfigured vertices, which is then given as input to phase plan (Section V). We note that, if no misconfigurations are found, control is given back to phase monitor for evidence collection. We present an example of verification for each property class in Section III.

Example 1: Let us consider a service certified for *data access level* at level 4 using RBAC. Each invocation to the *ToC* triggers the probe to check the metadata of the request, which includes information about the user making the request (e.g., authorization, role), and the operation to execute (data object to access, type of operation). All requests to be served by the *ToC* should be mediated by an RBAC system whose model requires data to be accessible for exceptional operations from authorized administrators only. We remark that the correctness of the specified policies is not in the scope of verification, as it is guaranteed by the certification authority specifying the constraints.

Let us then consider *data alteration detection* at level 3 requiring all data to be signed. The probe needs to check whether the storage supports signature functionalities and all stored documents are signed. Constraints can also specify the mechanisms (e.g., signature algorithms) to achieve the specified property and their correct configuration.

Let us finally consider *percentage of uptime*. The probe checks that the percentage of requests correctly served by the *ToC* is consistent with the value(s) specified in the property. Constraints can also specify the mechanism that must be used to achieve the specified property (e.g., a replica-based approach).

V. Adaptation Plan

The phase plan of the MAPE control loop process receives as input the list of misconfigurations calculated in phase analyze and returns as output a plan of adaptation for each of them, to maintain the certificate validity.

Adaptation activities need to keep track of existing configurations and their parameters to successfully adapt a misconfigured system. They also need to identify equivalent configurations that can be used as alternatives to an existing configuration. We use Feature Models (FMs), a formalism that has been heavily used in software product lines, as a way to model alternative configurations [11]. FMs can be applied in any domains to represent commonalities and differences in product/service configurations. They provide an overview of the configuration domain of a product and enable automated reasoning about features of interest. A feature model is formally defined as follows [12].

Definition 6 (Feature Model $\mathcal{FM}$): A feature model $\mathcal{FM}$ is a 6-tuple $\langle G, E_m, Gr_{xor}, Gr_{or}, C_{req}, C_{ex}\rangle$, where:

- $G(F, E, r)$ is a rooted tree with F as a finite set of features, $E \subseteq F \times F$ is a finite set of edges, $r \in F$ is the root feature;
- $E_m \subseteq E$ identifies the set of mandatory features;
- $Gr_{xor}, Gr_{or} \subseteq P(F) \times F$ represent alternative and optional feature groups, respectively. They are sets of pairs of child features together with their common parent feature and P defining the parameters of F.
- C_{req} and C_{ex} define finite sets of constraints specifying required and excluded features, respectively.

The feature model is used in this paper to represent alternatives between equivalent mechanisms (including specific configurations) that can be used to enforce certified properties. Figure 1 presents an example of *CIA* feature model generated following Definition 6.

Upon receiving the list of misconfigured vertices in the system model (phase analyze), phase plan initiates the adaptation process on the basis of the specified feature model. To interact with the feature model, a query needs to be executed using the information coming from phase analyze. A query is formally defined as follows.

Definition 7 (Query Q): Given a feature model $\mathcal{FM}$, a query Q over $\mathcal{FM}$ has the form $[pr/path]$ where pr identifies the property under verification and is further refined as $\hat{pr}/level$ according to property definition in Section II-B, and $path$ points to the mechanism and/or corresponding configuration in $\mathcal{FM}$ that show the misconfiguration, that is, for which corresponding constraints (Definition 2) in the system model are violated.

For instance, if a misconfiguration is found for property data alteration level 3 on the signature mechanism, we can define query [*integrity/data alteration detection/l3/signed data*]. Once the query is defined, it is given as input to phase plan that evaluates it. Three adaptation strategies can be executed as a result of the query evaluation.

A. Restore Configuration

Restore configuration strategy aims to restore all configurations back to the state that reflects the original certified configurations. This strategy has the advantage of restoring only the needed configurations (the misconfigured ones in the certification model), rather than instantiating the whole infrastructure and certification process from scratch.

Figure 2 shows the pseudocode of our planning algorithm, which receives as input a misconfiguration and returns as

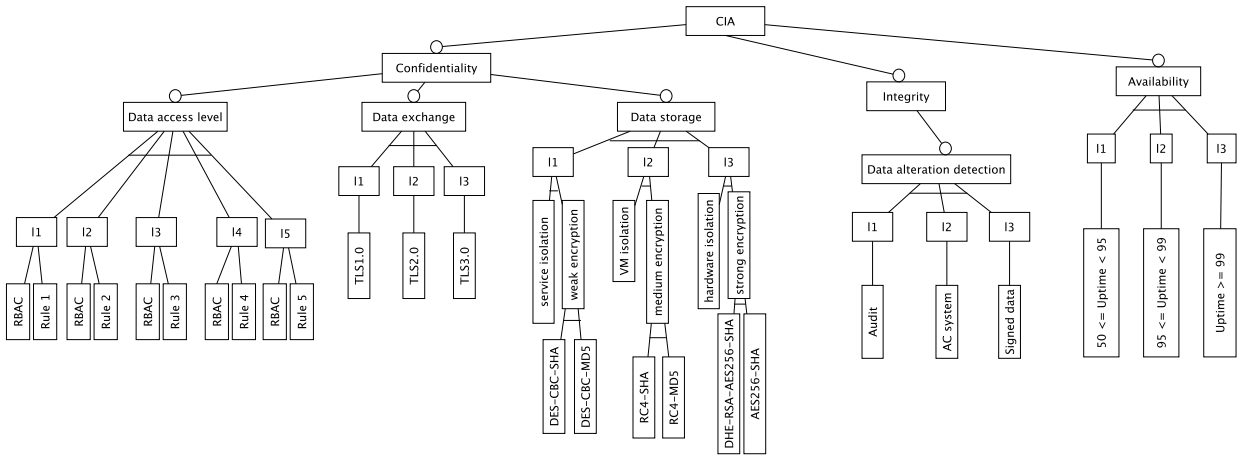

Figure 1. An example of feature model

output an adaptation plan. Restore configuration is considered first. Upon building a query Q to the feature model from the misconfiguration (function **Build_Query**), function **Get_Restore** is called with Q as parameter. This function checks if query Q is a valid query (function **Check**) or, in other words, whether the mechanism/configuration supporting the property still exist in the feature model. If a match is found, function **Get_Config** returns the configuration represented by the query. Otherwise, a null value is returned.

Example 2: Let us consider EHS in our reference scenario certified for property data alteration detection using *level 3* (signed data). At a certain time, the cloud provider moves EHS to a different environment, where mechanisms for data signature are not supported. Phase analyze raises a misconfiguration and then calls the restore configuration strategy, which checks whether the mechanism/configuration causing the misconfiguration is still in the feature model.

B. Equivalent Configuration

Restoring the certified configuration is not always possible due to changes in the provider infrastructure. Equivalent configuration strategy suggests an alternative cloud configuration that is equivalent to the certified one. An equivalent setting is therefore provided to maintain the same property, and in turn certificate validity. Formally, a configuration equivalence relation is defined as follows.

Definition 8 (Configuration equivalence relation $\equiv_{cf}$): Given a security property $pr \in \mathcal{P}$, two configurations $Cf_1, Cf_2 \in \mathcal{FM}$ are equivalent, denoted $Cf_1 \equiv_{cf} Cf_2$, *iff* security mechanisms $mec_1 \in Cf_1$ and $mec_2 \in Cf_2$ support property $pr \in \mathcal{P}$.

According to our algorithm in Figure 2, equivalent configuration is the second option to consider. Function **Get_Equivalent** implements the equivalent configuration approach. The function takes as input query Q and is recursively called to retrieve all equivalent configurations, moving up in the feature model tree (function **Parent**) until the property level is reached (function **Is_Child_Of**). For each execution of function **Get_Equivalent**, function **Get_Alternatives** is executed to retrieve all possible al-

```
INPUT
mc:=misconfiguration

OUTPUT
valid_configs:={configurations}

MAIN
Q:=Build_Query(mc);
valid_config:={};
valid_config∪:=Get_Restore(Q);
valid_config∪:=Get_Equivalent(Q);
valid_config∪:=Get_Restrictive([Q.p̂r/Q.level]);
}
return valid_config;

GET_RESTORE(Q)
if (Check(Q) == true)
  return Get_Config(Q);
else return {};

GET_EQUIVALENT(Q)
valid_alternatives_list:={};
if (Is_Child_Of(Q,Q.p̂r/Q.level)) {
  alternative_configs:=Get_Alternatives(Q);
  for each element el∈alternative_configs {
     if (Q≡cf el)
       valid_alternatives_list∪=el;
  }
  return valid_alternatives_list∪Get_Equivalent(Parent(Q));
}
else return {};

GET_RESTRICTIVE(Q)
valid_alternatives_list:={};
alternative_configs:=Get_Alternatives(Q);
for each element el∈alternative_configs {
     if (Q.level>cf el.level)
       valid_alternatives_list∪=el;
}
return valid_alternatives_list;
```

Figure 2. Planning algorithm

ternatives identified by the query. All alternatives are then checked (**for each** cycle) to find equivalent configurations on the basis of equivalence relation $\equiv_{cf}$ (Definition 8). Equivalent configurations are stored in a list and returned as the output of function **Get_Equivalent**.

Example 3: Let us consider EHS in our reference scenario certified for property confidentiality of data storage at *level 3*, with AES256-SHA algorithm. Upon EHS deployment, an inconsistency is detected: the in-production environment does not support encryption algorithm AES256-SHA. According to the feature model in Figure 1 and our algorithm in Figure 2, phase plan first checks alternatives for query [*confidentiality/data storage/l3/strong algorithm*] identifying DHE-RSA-AES256-SHA as a proper alternative to AES256-SHA. Since DHE-RSA-AES256-SHA is not available in current settings, phase plan moves up in the feature model tree and checks alternatives for query [*confidentiality/data storage/l3*]. It then identifies physical data storage isolation as a proper alternative to encryption for maintaining the property.

C. Restrictive Configuration

When no equivalent configurations can satisfy the certified property, a more restrictive (or stronger) configuration than the certified one is considered. Formally, a configuration restriction relation is defined as follows.

Definition 9 (Configuration restriction relation $>_{cf}$): A configuration $Cf_1 \in \mathcal{FM}$ is more restrictive than configuration $Cf_2 \in \mathcal{FM}$, denoted $Cf_1 >_{cf} Cf_2$, *iff* configurations Cf_1 and C_2 support the same property pr_i and security levels $l_n \in Cf_1$ and $l_m \in Cf_2$ are such that $l_n > l_m$, that is, l_n is more restrictive or stronger than l_m.

According to our algorithm in Figure 2, restrictive configuration is the last option to consider. Function **Get_Restrictive** implements the restrictive configuration approach. The function takes as input $Q.pr$ (i.e., [$\hat{pr}$/*level*]) and returns as output restrictive configurations. Function **Get_Restrictive** calls function **Get_Alternatives** to retrieve all possible alternatives based on $Q.pr$. All alternatives are then checked (**for each** cycle) to find restrictive configurations on the basis of restriction relation $>_{cf}$ (Definition 9). Restrictive configurations are stored in a list ordered by their level and returned as the output of function **Get_Restrictive**.

Example 4: Let us consider EHS in our reference scenario certified for property data exchange confidentiality at *level 2*, with an encryption algorithm of medium strength. However, by verifying the service at runtime, we discover that the encryption algorithm is not working properly because of a bug. Phase plan first checks whether there is an alternative algorithm at level 2, if not it checks whether a more restrictive configuration is available. In our example, mechanisms at *level 3* are considered, prescribing a strong encryption algorithm or hardware isolation.

VI. Adaptation Enforcement

The phase execute of the MAPE control loop enforces the adaptation strategies emerged during phase plan. It is performed by dedicated components that define the actions to take based on the selected strategy. Once the adaptation is committed successfully, phase execute passes the control to phase monitor to verify the status of the committed changes (MAPE control loop restarts). Phase execute receives as input a list of valid configurations: zero or one restore configuration, $[0, n]$ equivalent configurations, and $[0, m]$ restrictive configurations. It iterates over the list of valid configurations and finds the first configuration that satisfies the property under verification. We note that once one valid configuration is deployed, the process ends. We also note that the n alternative (the m restrictive, resp.) configurations are considered in the order returned by phase plan.

Phase execute is a 3-step process. The first step (*reset*) removes the existing mechanism causing misconfiguration. The second step (*setup*) instantiates the mechanism supporting the property under verification, by restoring the original configuration, or deploying an equivalent or restrictive configuration. The last step (*check*) verifies whether the selected mechanism has been successfully deployed and restarts MAPE control loop.

A. Integration with an Industrial Simulation Environment

We have simulated our approach in the British Telecommunications infrastructure based on Appcara Appstack [13]. Appcara Appstack is a cloud management layer that eliminates the traditional server/component template-based solutions, and uses a dynamic configuration modeling framework that captures all details about application configurations. All the details regarding deployed applications and their high-level and low-level configurations are stored in a repository using a patent-pending data model.

In the BT infrastructure [13], an agent-based automation system is used for monitoring specific pre-defined system properties (mainly availability and performance). Once a system property does not comply with the system requirements, the agent triggers an event. The event is received by the system administrator choosing a proper configuration to satisfy the property that triggered the event. Despite the fundamental advantages given by the current approach, it is affected by two main limitations: *i)* lack of full automation of the adaptation process, which could result in delays in responsiveness and affect the quality of user experience; *ii)* the administrator enforces adaptation recipes without evaluating the interference of the changes with other activities or configurations (e.g., a scale in the number of replicas can increase availability, reducing consistency).

The enhancement of this industrial solution with the functionality described in this paper and implementing the architecture in [8] can mitigate the above limitations. A solution relying on certification-based verification can reduce

Table II
POSSIBLE EHS ADAPTATIONS FOR PROPERTY DATA STORAGE

Adaptation strategy	Mechanisms
Restore configuration	data storage/l2/medium encryption/RC4-SHA
Equivalent configuration	data storage/l2/medium encryption/RC4-MD5
	data storage/l2/VM isolation
Restrictive configuration	data storage/l3/strong encryption/DHE-RSA-AES256-SHA
	data storage/l3/strong encryption/AES256-SHA
	data storage/l3/hardware isolation

the need of involving the administrator in the final adaptation decision. In such an enhancement, trust in decision making will be grounded in the trustworthiness of the certification authority, which drives alternative configuration selection. Also, our solution being based on MAPE control loop can continuously handle side effects introduced by deploying either automatically or semi-automatically a chosen configuration, and evaluate whether a decision invalidates a certificate, proposing possible alternative configurations.

Our approach can be integrated within the BT infrastructure by extending the logic of the continuous monitoring and policy enforcement layer with the capability to support the certification framework and collect evidence for a wider scope of activities. This will be complemented by extending the data/workload-driven approach of solutions such as Appcara Appstack and their recipe-based service management model to enforce the adaptation plans in Section V.

B. Walkthrough Example

We illustrate a walkthrough example describing how the different adaptation strategies are executed using tool FaMa-FW (http://www.isa.us.es/fama/) in the BT environment. We consider EHS system in our reference scenario deployed on top of Appcara Appstack and requirement R2, requiring all EHS components to store data in an encrypted form. Appstack data model includes all details about the used encryption algorithms, key lengths, and the encryption libraries. According to the feature model in Figure 1, we assume EHS system to be certified for property data storage confidentiality at level 2 (medium encryption) using RC4-SHA. Once the deployment is done, phase analyze reports an inconsistency: the encryption algorithm (RC4-SHA) defined in the data model is not working properly. At this point, the planning algorithm in phase plan (Figure 2) initiates the adaptation process by executing query [data storage/l2/medium encryption/RC4-SHA]. A list of possible adaptations of the EHS system is returned to maintain property data storage confidentiality (Table II).

Upon receiving the list of adaptations, phase execute sends an event to Appstack to replace mechanism RC4-SHA with an alternative option depending on the type of adopted adaptation (step *setup*), as follows.

Restore configuration. It aims to restore mechanism RC4-SHA by re-starting the encryption service with a new installation of RC4-SHA library. Appstack executes the change and if the misconfiguration persists, the equivalent configuration strategy takes over.

Equivalent configuration. According to the results of phase plan in Table II, we have two possible alternatives: *i)* to substitute RC4-SHA with an equivalent encryption algorithm, *ii)* to enforce data isolation by moving data to the storage of an isolated VM. Since both strategies have the same priority, phase execute randomly chooses one configuration. Appstack changes the data model by putting the mechanism in place, then the service behavior is checked to make sure that it is working properly. If both of the two configurations are not working properly, the restrictive configuration strategy is executed.

Restrictive configuration. According to the results of phase plan in Table II, we have three alternatives: *i)* to substitute RC4-SHA with DHE-RSA-AES256-SHA, *ii)* to substitute RC4-SHA with AES256-SHA, *iii)* to enforce physical data isolation by moving the storage to a different physical machine. All three alternatives, being at the same level, have the same priority and are chosen one-by-one until a working configuration is found or all possibilities have been evaluated.

If none of the aforementioned strategies is able to support the property under adaptation, meaning no alternatives are available to support the property, the appropriate action is taken (e.g., the certificate is revoked).

We note that our solution builds on already deployed frameworks, that is, the certification framework and the BT cloud management approach. The overhead introduced by our solution is therefore only in the calculation of alternative configurations in FM based on FaMa-FW tool. This calculation shows a negligible execution time of $66ms$ for 6×10^3 configurations.

VII. RELATED WORK

Cloud adaptation has been first addressed to deal with cloud migration. Jamshidi et al. [14] published a systematic literature review on studies focusing on planning, executing, and validating legacy systems migration to the cloud. More attention has been increasingly given to cloud self-adaptation to cope with runtime changes. The nature of the cloud allows different types of native adaptations which fall typically in two categories: *i)* services adaptation, where the deployed service is adapted to guarantee higher availability and fault tolerance (e.g., substitute services in an orchestration) [15], *ii)* configuration adaptation, which consists of adapting or reconfiguring the supporting infrastructure to guarantee the deployed service requirements. Garcia-Galan et al. [11] focused on automating the search of the most suitable configuration of a given IaaS provider using feature models. Schroeter et al. [16] used an extended feature model (EFM) to represent variability of functionality and service quality to handle runtime self-adaptive configurations. Pasquale et

al. [17] developed a user-centric adaptation approach which makes use of MAPE to support the adaptation process. Singh and Chana [2] proposed a systematic literature review with an overview of the state of the art in the field of autonomic cloud. The main idea of the review focuses on how services can self-manage their resources and their environment. From an industrial point of view, a special focus has been dedicated to cloud infrastructure adaptation in response to customers needs. However, most of the proposed solutions rely on a limited number of pre-defined properties to monitor and, in many cases, the adaptation options require manual intervention [13], [18]. Addressing a novel problem, our work is complementary to existing solutions for cloud adaptation and can be applied in conjunction with them. In particular, our solution can be applied in conjunction with both academic [3], [4] and industrial approaches [13], [18]–[20], providing two important extensions. First, an automatic approach based on certification; second, the adaptation of cloud configurations to maintain security properties.

VIII. Conclusions

Cloud computing supports an ecosystem subject to continuous context changes and unexpected events. In this scenario, guaranteeing a stable quality of service becomes a critical problem. In this paper, we presented a certification-based adaptation technique for cloud services, which maintains stable security behavior of cloud-based systems. The proposed approach relies on MAPE control loop process, and provides a transparent, user-informed adaptation based on reliable and trustworthy evidence coming from certification activities.

Acknowledgments

This work was partly supported by the project "A trustworthy certification approach for cloud-based composite services" under the program "Piano sostegno alla ricerca 2015" funded by Università degli Studi di Milano.

References

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

[2] S. Singh and I. Chana, "Qos-aware autonomic resource management in cloud computing: A systematic review," *ACM CSUR*, vol. 48, no. 3, p. 42, 2015.

[3] C. Inzinger, B. Satzger, P. Leitner, W. Hummer, and S. Dustdar, "Model-based adaptation of cloud computing applications." in *Proc. of MODELSWARD 2013*, Barcelona, Spain, February 2013.

[4] S. Farokhi, P. Jamshidi, I. Brandic, and E. Elmroth, "Self-adaptation challenges for cloud-based applications: a control theoretic perspective," in *Proc. of Feedback Computing 2015*, Seattle, USA, April 2015.

[5] 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.

[6] B. Jacob, R. Lanyon-Hogg, D. K. Nadgir, and A. F. Yassin, *A practical guide to the IBM autonomic computing toolkit*, IBM, 2004.

[7] C. Consortium, *D2.1: Security-aware SLA specification language and cloud security dependency model*, http://cumulus-project.eu/index.php/public-deliverables, Accessed in March 2016.

[8] C. Ardagna, R. Asal, E. Damiani, and Q. Vu, "On the management of cloud non-functional properties: The cloud transparency toolkit," in *Proc. of NTMS 2014*, Dubai, UAE, March-April 2014.

[9] S. Tilley and T. Parveen, *Software testing in the cloud: migration and execution*. Springer, 2012.

[10] M. Anisetti, C. Ardagna, E. Damiani, and N. El Ioini, "Trustworthy cloud certification: A model-based approach," in *Proc. of SIMPDA 2014*, Milan, Italy, November 2014.

[11] J. García-Galán, P. Trinidad, O. F. Rana, and A. Ruiz-Cortés, "Automated configuration support for infrastructure migration to the cloud," *FGCS*, vol. 55, pp. 200–212, 2016.

[12] V. Štuikys, *Smart Learning Objects for Smart Education in Computer Science: Theory, Methodology and Robot-Based Implementation*. Springer, 2015, ch. Background to Design Smart LOs and Supporting Tools, pp. 185–209.

[13] Appcara, http://www.appcara.com/products/appstack-r3, Accessed in March 2016.

[14] P. Jamshidi, A. Ahmad, and C. Pahl, "Cloud migration research: a systematic review," *IEEE TCC*, vol. 1, no. 2, pp. 142–157, 2013.

[15] E. Cavalcante, T. Batista, F. Lopes, A. Almeida, A. L. de Moura, N. Rodriguez, G. Alves, F. Delicato, and P. Pires, "Autonomous adaptation of cloud applications," in *Proc. of DAIS 2013*, Florence, Italy, June 2013.

[16] J. Schroeter, P. Mucha, M. Muth, K. Jugel, and M. Lochau, "Dynamic configuration management of cloud-based applications," in *Proc. of SPLC 2012*, Salvador, Brazil, September 2012.

[17] J. García-galán, L. Pasquale, P. Trinidad, and A. Ruiz-Cortés, "User-centric adaptation analysis of multi-tenant services," *ACM TAAS*, vol. 10, no. 4, p. 24, 2016.

[18] VMware, "Multi-cloud environments are becoming the new normal for it," http://tinyurl.com/htse8ol, Accessed in March 2016.

[19] J. Daniel, F. El-Moussa, G. Ducatel, P. Pawar, A. Sajjad, R. Rowlingson, and T. Dimitrakos, "Integrating security services in cloud service stores," in *Trust Management IX*. Springer, 2015, pp. 226–239.

[20] J. Daniel, T. Dimitrakos, F. El-Moussa, G. Ducatel, P. Pawar, and A. Sajjad, "Seamless enablement of intelligent protection for enterprise cloud applications through service store," in *Proc. of CloudCom 2014*, Singapore, December 2014.