# Incremental Certification of Cloud Services

Maria Krotsiani, George Spanoudakis, Khaled Mahbub
School of Informatics
City University London, UK
{Maria.Krotsiani.1, G.E.Spanoudakis, K.Mahbub}@city.ac.uk

***Abstract*—Cloud is becoming fast a critical infrastructure. However, several recent incidents regarding the security of cloud services clearly demonstrate that security rightly remains one of the major concerns of enterprises and the general public regarding the use of the cloud. Despite advancements of research related to cloud security, we are still not in a position to provide a systematic assessment of cloud security based on real operational evidence. As a step towards addressing this problem, in this paper, we propose a novel approach for certifying the security of cloud services. Our approach is based on the incremental certification of security properties for different types of cloud services, including IaaS, PaaS and SaaS services, based on operational evidence from the provision of such services gathered through continuous monitoring. An initial implementation of this approach is presented.**

***Keywords–cloud services; security certification; continuous monitoring***

## I. INTRODUCTION

Cloud technology offers the opportunity for utilising computational capabilities offered as computation, data, storage, and network cloud services upon demand without owning the resources that realise and offer them [26]. The use of cloud services is spreading fast, formulating a market that is expected to reach a value of $66bn by 2016 [29]. Despite their fast spread, the use of cloud services has been associated with several security problems. These include breaches of integrity, confidentiality and privacy of customer data on clouds [6][9][18], spamming, wrapping and cross-site scripting attacks [6], various forms of Denial-of-Service (DoS) attacks resulting in reduced application and data availability [9][24], and Authentication, Authorization and Accounting (AAA) vulnerabilities [9][18]. Thus, the use of the cloud has created significant concerns regarding the security of the data and services offered through it [6][9][18]. Frequent reports of cloud security incidents spanning across major providers indicate that these are valid concerns [14].

The provision and assessment of cloud service security is difficult and not well supported by the existing state of the art. Researchers have made significant progress delivering methods and tools for access control and identity management on clouds [1], secure storage protocols (e.g., proof-of-retrievability [3], proof-of-storage protocols [31]), encryption and key management [17], and secure virtualization [12]. Despite this work, the security of cloud services is far from perfect. This is due to several vulnerabilities of cloud service provision that are related to the possibility of breaches of integrity, confidentiality and privacy due to multi-tenancy of services; interference between security mechanisms operating at different layers of cloud services (infrastructure, platform and software services); interference between security and cloud virtualization/optimisation mechanisms (e.g., spreading of DoS attacks due to load balancing in cloud federations [24]); and subverting administrative functions of the cloud infrastructure.

The risk arising from these vulnerabilities is increased by dependencies between services at all the layers of the cloud stack (i.e., IaaS, PaaS and SaaS services), which may also evolve dynamically (e.g., changing VMs, configurations of platform services, or deployed software services). These dependencies and their dynamic changes make it difficult to introduce appropriate pre-deployment and operation controls for assessing and guaranteeing the security [9][24]. Furthermore, the exact provision of cloud services is frequently opaque, making it difficult to assess cloud security through audit mechanisms.

Under these circumstances, the security and trustworthiness of cloud services require continuous and transparent assessment. Certification has a long history as a mechanism for verifying properties of software systems and increasing trust in them, mainly due to the liability that it entails for the authority that issues the certificates. However, existing certification methods, such as those based on the Common Criteria model [10], focus mainly on systems with a stable structure that operate under stable operational conditions and, therefore, they are not suitable for the cloud. Recent research focuses on certification of service-based systems, whose structure can change dynamically [22]. However, it still ignores the deployment infrastructure and platform services that are involved in the provision of software services in a cloud. Also, SOA certification research [22][23] is centered on certificates produced through pre-deployment testing and/or formal analysis of services without incorporating real and continuous cloud service monitoring data.

In this paper, we propose a novel cloud service certification approach that addresses this gap. Our approach is based on the creation of incremental certificates for security properties of cloud services. In such certificates, the evidence required for assessing and verifying security properties is acquired through continuous monitoring of the operation of cloud services. Hence, the evidential basis for the assessment of properties can cover contextual conditions that might not be possible to envisage, test or simulate through other forms of assessment (e.g., testing and static

 ISBN: 978-1-61208-298-1 

analysis) before deploying a cloud service. Continuous monitoring can capture contextual conditions in cloud service provision as, for example, changes in the population of co-tenant services, the deployed virtualization and optimisation strategies and mechanisms, and network and middleware configurations in a cloud, which are difficult to take into account in static forms of assessment. It can also capture and adapt to the migration of SaaS cloud services within cloud federations, providing support for the adaptation of monitoring infrastructures when this happens.

The rest of this paper is organized as follows. Section II overviews our approach. Section III introduces the certification models that drive the incremental certification process. Section IV details the certification process, gives examples of how it is carried out, and overviews the tool support for our approach. Section V overviews related work, and finally, Section VI provides concluding remarks and directions for future work.

## II. OVERVIEW OF INCREMENTAL CERTIFICATION

### A. Certificates and certification models

The core model of certificates that can be issued in our approach is shown in Figure 1. As the model shows, a certificate asserts a *security property* regarding an *asset* of a cloud service (e.g., a specific service operation, a set of service operations, data managed by the service). Certificates are produced based on a *certification model* that is defined (or endorsed) by a *certification authority*, and are signed by this authority. The certification model is based on an *assessment scheme* determining what evidence should be taken into account for assessing a property. The assessment scheme also determines how frequently the evidence should be checked and when the accumulated evidence will be sufficient for issuing the certificate through an *evidence aggregation scheme*.

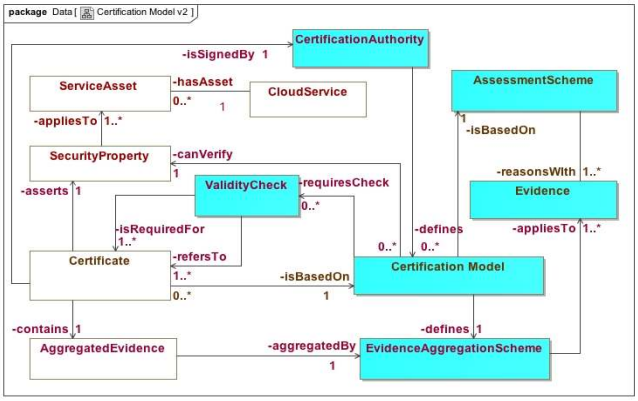


Figure 1: Model Based Certificates

The certification model defines also *validity checks* that should be executed before issuing a certificate. Such checks may, for example, require that the monitoring infrastructure, which is used to gather the operational evidence for the certificate C, has itself a certificate C' confirming the integrity and non-repudiation of the monitoring data that it captures and provides. Also, C' had to be valid for the entire period over which the particular monitoring infrastructure has been used to gather evidence for C.

When the evidence gathered for a certificate type is sufficient for verifying the security property related to it (as determined by the certification model), a certificate that is an instance of this type could be issued. However, even after they are issued, certificates can be subject to changes in the operational conditions of the cloud service that they are associated with. The possible updates and other key changes in the life cycle of incremental certificates are described by a generic life cycle model that we discuss next.

### B. Lifecycle of incremental certificates

The life cycle of incremental certificates is described by the UML statechart diagram of Figure 2. The first state in the certificate's lifecycle is *Activated*. In this state, a certificate type is activated with a unique ID; a reference to the certification model that will be used for its incremental assessment and for issuing and updating certificates that are instances of it; a reference to the asset(s) of the cloud service that it refers to; and a digital signature of its issuer. The signature confirms that certificates, which are instances of the particular type, may be potentially issued for the referenced cloud service, based on the specific certification model and the evidence that needs to be collected.

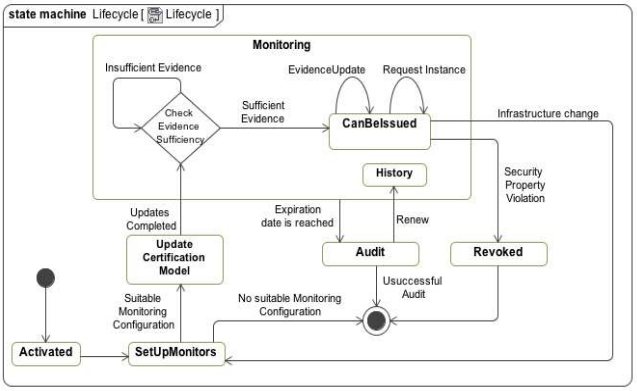


Figure 2: Life cycle of incremental certificates

After being activated, a certificate type enters the *SetUpMonitors* state. At this state, a monitoring infrastructure that will be used to acquire the operational evidence required for the assessment of the certificate is set up. If a suitable monitoring infrastructure for the type can be identified and set up, the certificate type moves to *UpdateCertificationModel* state. At this state, a concrete operational specification of the security property of the certificate type is generated for the particular infrastructure. This specification will enable the monitoring infrastructure to determine the monitoring events (evidence) required for the assessment of the property and how to use them in order to assess the property. When these updates are completed and recorded in the certification model, the certificate type moves to *Monitoring* state. Note, however, that if no suitable monitoring configuration is found, the certificate type will cease to exist.

Whilst at the monitoring state, the evidence required for the assessment of the certificate type is continually gathered

 ISBN: 978-1-61208-298-1 

by the monitoring infrastructure. When the accumulated evidence becomes sufficient for confirming the validity of the security property of the type, the certificate type gets to the sub state *CanBeIssued*. This state indicates that certificates of the particular type can be issued by the certification platform. Whilst a certificate type is at the state *CanBeIssued*, its monitoring continues and, at specific checkpoints defined by the certification model, any additional operational evidence that is acquired for the type by the monitoring infrastructure is recorded in an aggregated format within the certification infrastructure (see the self-transition *EvidenceUpdate*).

When a certificate type gets to the state *CanBeIssued*, any agent interested in it (whether an application or a human actor) can request the generation of a certificate that is an instance of the particular type from the certification infrastructure by using the certificate type's unique ID. Upon such a request, the certification infrastructure will check if the extra validity conditions for the certificate type (if any) are satisfied and, if they are, it will generate a certificate instance and return it back to the requester. The generated instance will be identified by the combination of the ID of the certificate type and its own unique ID. The certificate instance will also record the expiration date of its type (as this date will apply to it as well). These actions are executed during the transition *RequestInstance* in Figure 2. The issued instance of the certificate type will incorporate the aggregated evidence until the last checkpoint when monitoring data were retrieved from the monitoring infrastructure and aggregated for the certificate type. It will also contain the timestamp of the earliest and latest available evidence for the security property that it asserts, in order for its users to know the exact period covered by the evidence.

A certificate type that is at the state *CanBeIssued* may also be revoked. This will happen if the monitoring infrastructure identifies new monitoring data contradicting the evidence gathered so far and indicating that the relevant security property has been compromised. The occurrence of such evidence is signified by the transition *Security Property Violation* to the state *Revoked*. When a certificate type gets to the state *Revoked*, all the previously generated instances of it will be revoked. Also, the certificate type will be flagged as revoked in the certification infrastructure in order to prevent the generation of new instances of it, and its monitoring will be terminated.

In line with traditional models of certification, a certificate type has an expiration date. When this date is reached, the certificate type will move to the state *Audit*. This state signifies an audit of the certification model and the evidence gathered for the certificate type. The certification authority, which defined the certification model, will carry this out in order to ascertain that it may continue to use the certification model for producing certificates. If the audit is successful, the certificate type can be renewed: the transition *Renewed* brings the certificate back to the state where it was prior to reaching its expiration date and moving to the state *Audit*. If the audit is unsuccessful, the specific certificate type will be terminated. In this case, a notification should also be sent to owners of any issued instance certificates of the type to notify them that the type no longer exists but that the instances of it that have been issued will remain valid until their expiration date.

Following the expiration of a certificate instance, its agent can request from the certification infrastructure to provide a renewed instance of the same certificate type. The infrastructure will be able to do so only if the certificate type at this stage is at the state *CanBeIssued*.

The life cycle mode also reflects reactions to changes related to the cloud infrastructure where the certified service has been deployed, or to the monitoring infrastructure that is used to generate the operational evidence for the certificate. Such changes may require a change in the monitoring infrastructure that is used to gather the operational evidence underpinning the certificates of the given type and the concrete operational specification of the security property of the certificate type that drives the operation of the monitoring infrastructure. The evidence, however, that has been held so far in the certification infrastructure regarding the property may (depending on the certification model) remain relevant and should be maintained by the certification structure so that to be used when issuing new instances of the given type.

Changes in the cloud infrastructure and/or the monitoring infrastructure will trigger the transition of the certificate type to the *SetUpMonitors* state to check whether after the changes, the cloud deployment infrastructure of the service provides the monitoring capabilities required for continuing the monitoring of the specific certificate type. If successful, the certificate will move to the *UpdateCertificationModel* during which the certification model will be updated as described previously. When this completes successfully, the certificate will transit back to the *evidence* accumulation state inside *Monitoring*. If, however, the security property cannot be monitored after the changes, the certificate type will cease to exist.

### C. Incremental Certification Infrastructure

The generation of incremental certificates is supported by a proof-of-concept certification infrastructure (CeRTiN). The architecture of CeRTiN is shown in Figure 3.

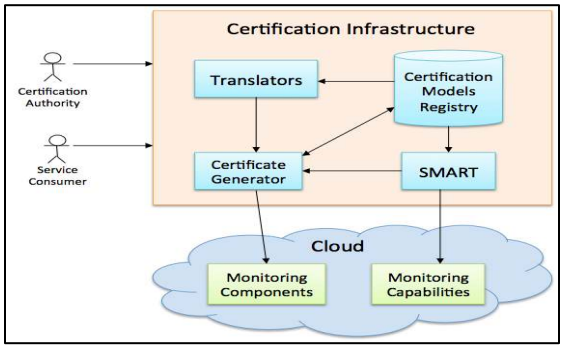


Figure 3: Certification Infrastructure

CeRTiN consists of a *Certificate Generator (CG)*, a *service monitorability reporting tool (SMART)*, a set of *translators*, and a *certification models registry*. The certificate generator has the responsibility for initiating and

managing the process of creating incremental certificates following the life cycle model introduced in Sect. II.B, upon requests for certification authorities. The operation of the certificate generator is driven by certification models, maintained as part of a model registry. To realise the functionality underpinning the state *SetUpMonitors* and *Monitoring* in the life cycle model, the generator interacts with *SMaRT* and *Translators*, and *Monitoring components* on cloud platforms, respectively. *SMaRT* has responsibility for checking (i) whether the cloud on which the cloud service to be certified is deployed has the monitors required for the incremental certification of the service, and (ii) for assembling an appropriate monitoring platform. To do so, *SMaRT* has access to the certification model to be applied and descriptions of the monitoring capabilities available in cloud platforms. *Translators* have the responsibility for translating the security property to be verified by certificates into an operational monitoring specification for the monitoring platform assembled by SMaRT. Finally, the monitoring components on cloud platforms have the responsibility for providing the raw monitoring evidence from cloud monitoring components.

## III. SPECIFICATION OF CERTIFICATION MODELS

As discussed in Sect. II, the generation of incremental certificates is driven by certification models. In this section, we define the language for specifying such models as an XML schema. The top-level structure of this schema is shown in Figure 4 and the elements at this level of the schema are discussed below.

### A. Certification model elements

*CASignature*: The element *CASignature* represents the digital signature of the certification authority that has defined/advocated the certification model.

*AbstractSecurityProperty*: This element defines the security property that is to be certified by the particular certification model. The security property must be defined in an abstract form that is independent from the language used by the monitoring infrastructure that will be used for gathering operational evidence for the property. This is because the monitoring infrastructure that will be used for different certificates of the model may vary across different cloud platforms. Also, the monitoring infrastructure may change during the lifecycle of the certificate type when, for example, the service or the constituent services and components that are the subject to incremental certification migrate across different clouds (e.g., within a cloud federation). During the monitoring infrastructure set up state in the certificate type life cycle model, the abstract property is translated automatically to the concrete property that will then be used to drive the operational monitoring (see below).

In the current implementation of our approach, the abstract security property is expressed using a subset of the language for specifying Service Level Agreements (SLAs) that was introduced by the SLA@SOI project, known as SLA* model [19]. In particular, we are using the part of the language that enables the definition of guaranteed terms in an SLA. SLA* has been chosen for two reasons. The first is that it defines several build-in security properties such as integrity, non-repudiation, availability and forms of confidentiality as "standard" guarantee terms. The second reason is that it provides the syntactic and semantic means for customizing the definitions of built-in properties and/or defining new properties.

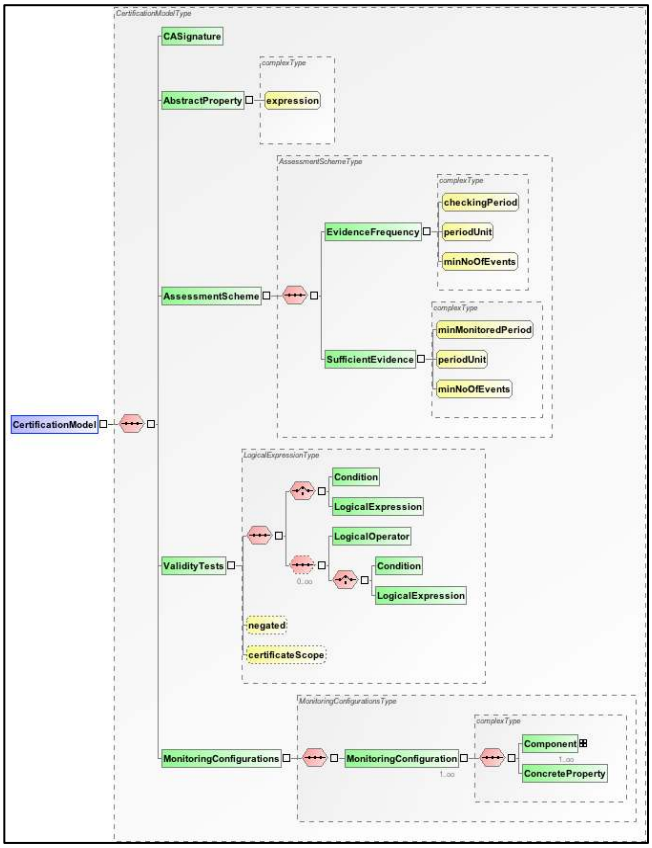


Figure 4: Certification model schema (top level)

The certification model schema provides two options for specifying abstract security properties in SLA*: (1) to specify the property as a string, according to the BNF syntax of SLA*, or (2) to use the XML schema defined for SLA*. A full account of SLA* is beyond the scope of this paper and can be found in [19]. In Figure 5, however, we show an example of specifying the availability of a service as defined in SLA*.

*AssessmentScheme:* This element defines conditions regarding the assessment of the evidence that should be satisfied, in addition to the adherence of the cloud service to the security property, for the certificate to become issuable. These conditions relate to the frequency and sufficiency of evidence collection, and are specified by the following sub-elements of an assessment scheme:

- *EvidenceFrequency* – This element defines the *checkingPeriod*, which states how often the events should be checked, and/or the *minNoOfEvents*, which declares the minimum number of events that should occur in a specific period of time.

- *SufficientEvidence* – This element defines sufficiency conditions regarding the extent of collected evidence for issuing a certificate. Such conditions are specified by the sub-elements of *SufficientEvidence*, namely:
  - *minMonitoredPeriod* that states the minimum time needed for monitoring,
  - *minNoOfEvents* that states the minimum number of events (evidence) that need to be monitored.

*Validity tests*: A certification model may, in addition to the assessment scheme, define extra validity tests as preconditions for issuing a certificate of a given type. These tests may relate, for example, to conditions regarding the cloud where the service is deployed (e.g., requiring that the cloud offers full isolation of virtual machines) or the adherence of other services that this service may depend on to standards (e.g., requiring that a storage service, which is used by a SaaS service implements correctly a proof-of-retrievability protocol [3]) or the monitoring infrastructure itself (e.g., requiring the integrity of the transmission of monitoring events and results inside the infrastructure and to external clients of it). Such conditions are specified by the element *validityTests* in the certification model schema. Our approach to the specification of validity tests assumes that any related components will have other certificates confirming their adherence to the required conditions and therefore the validity tests are expressed as logical conditions against the contents of such certificates.

*MonitoringConfigurations*: This element specifies the list of the monitoring configurations that have been used to collect the evidence for generating certificates. Each monitoring configuration includes:

- A list of *components* of the monitoring environment. These components can be of two types: (1) *sensors,* which are components capable of capturing and transmitting primitive monitoring events, and (2) *reasoners,* which are components capable of analyzing events and checking whether monitoring conditions are satisfied (aka *monitors*). The *start* and *end* of each time period during which it has been used for collecting evidence for the specific certification model
- *ConcreteSecurityProperty* – This element provides the concrete operational specification of the security property that is to be certified by the model, expressed in the language accepted by the reasoner(s) of the particular monitoring configuration. The concrete security property is generated automatically from the abstract security property once a monitoring configuration is selected as we discuss in Sect. IV.C.

### B. Example of Certification Model for Availability Property

Figure 5 presents an example of a certification model for the availability of a service, i.e., the ratio of the period of time during which a service cannot respond to calls of its operations without producing an error (i.e., it is unavailable), over the total period of time during which the service is deployed.

As shown in the figure, the value of the *expression* attribute in the *AbstractSecurityElement* is set to *availability* (i.e., a standard term of SLA* with the meaning defined above). The element *AssessmentScheme* of the model specifies that monitoring should be performed every 24 hours for minimum of 1000 events in this period (see the attribute values of *EvidenceFrequency* element) and monitoring should be performed for at least 30 days for minimum 100000 events in this period in order to issue a certificate (see the attribute values of *SufficientEvidence* element). The *ValidityTest* element specifies a precondition that should be met to issue a certificate. The precondition in this example specifies that the *integrity* of the transmission of monitoring events and results should be maintained. Finally, in the *MonitoringConfiguration* element, it is specified the monitoring environment that will be used to monitor the events and collect the evidence.

```
<ns1:CertificationModel  xmlns:xsi='http://www.w3.org/2001/XMLSchema-
instance'
  xmlns:ns3='http://slasoi.org/monitoring/citymonitor/xmlrule'
  xmlns:ns2='http://assert4soa.eu/schema/Assert_SQL'
  xmlns:ns1='http://www.cumulus.org/certificate/model'
  xsi:schemaLocation='http://www.cumulus.org/certificate/model
CertificationModel-v2.xsd'>
   <CASignature></CASignature>
   <AbstractSecurityProperty
      expression="http://www.slaatsoi.org/commonTerms#availability"/>
   <AssessmentScheme>
      <EvidenceFrequency checkingPeriod="24" periodUnit="hours"
        minNoOfEvents ="1000"></EvidenceFrequency>
      <SufficientEvidence minMonitoredPeriod="30" periodUnit="days"
           minNoOfEvents="100000"></SufficientEvidence>
   </AssessmentScheme>
   <ValidityTests negated="false" certificateScope="SINGLE">
      <ns2:Condition negated="false" relation="EQUAL-TO">
         <ns2:Operand1>
            <ns2:AssertOperand facetName="Assert" facetType="Assert">
              //ASSERTCore/SecurityProperty/@PropertyAbstractCategory
            </ns2:AssertOperand>
         </ns2:Operand1>
         <ns2:Operand2>
            <ns2:Constant type="STRING">
              http://www.assert4soa.eu/ontology/security/security#Integrity
            </ns2:Constant>
         </ns2:Operand2>
      </ns2:Condition>
   </ValidityTests>
   <MonitoringConfigurations>
      <MonitoringConfiguration>
         <Component type="REASONER">
            <EndPoint>http://localhost:8888/...</EndPoint>
         </Component>
         … … … …
      </MonitoringConfiguration>
   </MonitoringConfigurations>
</ns1:CertificationModel>
```

Figure 5: Example Certification Model

## IV. CERTIFICATION PROCESS

In the following, we give an example demonstrating the certification process based on the availability property specified in the certification model of Figure 5.

### A. Initiation of certification process

The certification process starts when a service provider makes a request to a Certification Authority (CA), in order to certify a security property for a service. The CA may use an existing Certification Model (CM) if it is suitable for the property and the service, create a new model for it, or reject the request. If a CM is identified, CA submits it to the Certificate Generator CG). CG has then to select and

configure a monitoring infrastructure for starting the incremental certification process.

### B. Selecting and configuring the infrastructure

Assuming that the request for certification is about the availability of a service S, the CM of Figure 5 could be selected and used for the certification of S. After identifying it, CG calls SMART (see Figure 3) to find if there is a monitoring infrastructure available in the cloud platform where the service is deployed that could monitor availability.

To do so, SMART parses the security property in the certificate model and generates an Abstract Syntax Tree (AST) of the property based on the BNF grammar of the security property language. Then, it searches through the registry of the monitoring capabilities of the cloud infrastructure to check if there are suitable sensors for providing the events and analysis functions required for monitoring the availability property.

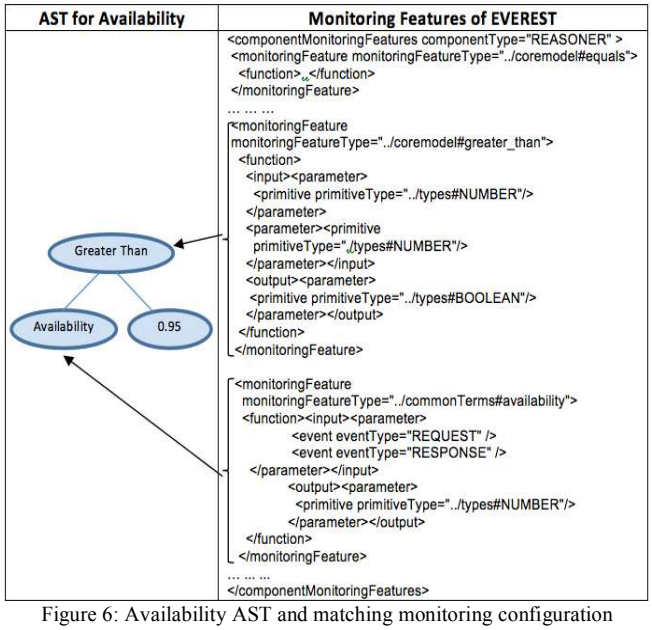


Figure 6: Availability AST and matching monitoring configuration

The capabilities of the monitoring components that may be available in the infrastructure are described according to the monitoring capability model introduced in [16]. According to this model, a monitoring component in a cloud infrastructure is described by its unique identifier within the infrastructure (uuid), its type (i.e., SENSOR or REASONER depending on whether the component can capture and transmit events or can analyse them to check if a monitoring condition is satisfied, respectively), and a list of *MonitoringFeatures*. The latter describe either *basic features* such as event capturing and transmitting operations (e.g., service operation requests and responses) or more complex computational *functions* (e.g., computation of average time between service calls, CPU load etc.).

SMART uses the AST for the property to find monitoring components, which have monitoring features matching each node of the tree. More specifically, a non-leaf node of AST would have the form *(Parent: Operator (OP), Children: left-hand side (LHS), right-hand side (RHS))* and each of OP, LHS and RHS needs to be matched with a monitoring component with appropriate features. A full match of the AST with monitors constitutes a candidate configuration. For these, SMART also checks if they satisfy any validity conditions for monitors that may have been defined in the certification model, and maintains only the configurations that are compliant. If there is more than one such configuration, it also performs a selection. Figure 6 shows the AST for availability and an example of a matching monitoring configuration.

### C. Deriving the monitoring specification

Following, the selection of a monitoring configuration, CG translates the abstract security property of the certification model into the monitoring language accepted by the REASONER component in the configuration.

In the current implementation of CeRTiN, we are using EVEREST (EVEnt REaSoning Toolkit [15]) to perform the monitoring required during the certification process. EVEREST is an open-source monitoring framework developed by the last two authors of this paper to support the monitoring of service-based systems. EVEREST supports the monitoring of properties expressed in *EC-Assertion*, a first order temporal logic language based on Event Calculus.

*EC-Assertion* specifies monitorable properties in terms of *events* and *fluents*. An event is something that occurs at a specific instance of time and has instantaneous duration. Fluent represent system states and are initiated and terminated by events. The basic predicates used by EC-Assertion for expressing events and fluents, and their meanings are summarized in Table I.

TABLE I: EC-ASSERTION PREDICATES

| Predicate | Meaning |
|---|---|
| *Happens(e,t∈ℜ(t$^L$,t$^U$))* | An event *e* of instantaneous durations occurs at some time point *t* within the time range $\Re(t^L,t^U)$ ($\Re(t^L,t^U)$=[ $t^L$,$t^U$]). |
| *HoldsAt(f,t)* | A state (aka *fluent*) *f* holds at time *t*. This is a derived predicate that is true if the *f* has been initiated by some event at some time point t' before t and has not been terminated by any other event within the range [t',t]. |
| *Initiates(e,f,t)* | Fluent *f* is initiated by an event *e* at time *t* |
| *Terminates(e,f,t)* | Fluent *f* is initiated by an event *e* at time *t* |
| *Initially(f)* | Fluent *f* holds at the start of system operation. |
| <rel>(x,y), <rel>::= = \| < \| > \| ≤ \| ≥ \| ≠ | The relation <rel> holds between the x and y. |

Based on these predicates, EC-Assertion expresses monitorable properties as *monitoring rules* of the form *body* ⇒ *head*. The meaning of a monitoring rule is that if its body evaluates to *True*, its head must also evaluate to *True*. EC-Assertion also uses *monitoring assumptions*, which have the same form as rules but their meaning is that when their body evaluates to *True*, their head can be deduced. Thus assumptions are used to deduce and/or record information about the state of the system during monitoring.

Operational monitoring specifications in EC-Assertion are produced from the specification of an abstract security property in a certification model by transforming the AST of the property into EC-Assertion formulas. The translation process is based on the use of predefined parametric

 ISBN: 978-1-61208-298-1 

monitoring templates for the standard terms of the SLA* model. These templates are retrieved and instantiated during the translation process when the relevant standard term of SLA* is encountered in a node of the AST generated for a security property. The details of this translation process are beyond the scope of this paper and can be found in [20].

Table II shows the parametric template used for the standard *Availability* property in SLA*. As discussed in Sect. III.B, service availability is defined as the ratio of the period during which a service is unavailable over the total period of monitoring a service.

TABLE II: AVAILABILITY TEMPLATE (EC-ASSERTION)

⟨*Availability*⟩$_{def}$==

**A0.Availability.<CaseId>: Initially**(LastServiceMonitoringPeriod(<_SrvId>, systemTime())

**A1.Availability.<CaseId>:**
**Initially**(UnavailablePeriods(<_SrvId> ,_PN, _P[]))

**A2.Availability.<CaseId>:**
**Happens**(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), t1, [t1,t1]) ∧
¬**Happens**(e(_id2, <_SrvId>, _Snd, Response(_O), <_SrvId>), t2, [t1,t1+<D>]) ∧ ¬∃ _PN, _ST,: **HoldsAt**(Unavailable(_PN, <_SrvId>, _ST), t1)) ∧
∃ _PN, _P[]: **HoldsAt**(UnavailablePeriods(<_SrvId>, _PN, _P[]), t1)) ⇒
**Initiates**(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), Unavailable(_PN+1, <_SrvId>, t1), t1) ∧ **Terminates(**e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), UnavailablePeriods(<_SrvId>, _PN, _P[]), t1) ∧
**Initiates(**e(_id1,_Snd,<_SrvId>,Call(_O),<_SrvId>),UnavailablePeriods(<_SrvId>,_PN+1,_P[]), t1)

**A3.Availability.<CaseId>**:
**Happens**(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), t1, [t1,t1]) ∧
**Happens**(e(_id2, <_SrvId>, _Snd, Response(_O), <_SrvId>), t2, [t1,t1+<D>]) ∧∃ _PNum, _ST: **HoldsAt**(Unavailable(_PN, <_SrvId>, _ST), t1) ⇒
**Terminates**(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), Unavailable(_PN, <_SrvId>, _ST), t1+1)

**A4.Availability.<CaseId>:**
**Happens**(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), t1, [t1,t1]) ∧
**Happens**(e(_id2, <_SrvId>, _Snd, Response(_O), <_SrvId>), t2, [t1,t1+<D>]) ∧ ∃ _PN, _ST,: **HoldsAt**(Unavailable(_PN, <_SrvId>, _ST), t1)) ∧
∃ _PN, _P[]: **HoldsAt**(UnavailablePeriods(<_SrvId>, _PN, _P[]), t2)) ⇒
**Terminates(**e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), UnavailablePeriods(<_SrvId>, _PN, _P[]), t2) ∧
**Initiates(**e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), UnavailablePeriods(<_SrvId>, _PN, append(_P[], t1 – ST)), t2)

**R.Availability.<CaseId>:**
**Happens**(e(_id1, _Snd, <_SrvId>, Call(_O), <_SrvId>), t1, [t1,t1]) ∧
**Happens**(e(_id2, <_SrvId>, _Snd, Response(_O), <_SrvId>), t2, [t1,t1+<D>]) ∧∃ _PN, _ST, _P []: **HoldsAt**(Unavailable(_PN, <_SrvId>, _ST), t1)) ∧
**HoldsAt**(UnavailablePeriods(<_SrvId>, _PN, _P[]), t2) ∧
**HoldsAt**(LastServiceMonitoringPeriod(<_SrvId>, _lmsTime), t2)) ⇒
sum(_P[]) / (t2 – _lmsTime) > K

The parametric availability template uses three fluents specific to availability computation. The fluent Unavailable keeps track of unavailability. The fluent has three parameters: (i) _PN that counts the number of unavailable periods for a monitored service; (ii) _SrvId that records the unique ID of the monitored service, and (iii) _ST that records the time point when the service becomes unavailable. To keep track of unavailable periods the template uses the fluent UnavailablePeriods. This fluent has also three parameters: (i) _SrvId that records the unique ID of the monitored service; (ii) _PN that records the count of unavailable periods of the monitored service; and (iii) _P[] that records duration of each unavailable period of the monitored service. The third fluent in the template, i.e., LastMonitoringPeriod, is used to record the starting time point of the monitoring session. This fluent has two parameters. The first parameter (i.e. <_SrvId>) records the unique ID of the monitored service, and the second parameter (i.e. systemTime()) signifies a system call executed by the monitor to obtain the current time of the system where the monitoring service is running.

The assumptions A0 and A1 initiate the *LastMonitoringPeriod* and *UnavailablePeriods* fluents respectively for first time. A2 starts new period of unavailability when a non-served event occurs and increases the number of unavailability periods. A3 terminates a current period of unavailability for a service. The assumption A4 records the length of a terminated period of unavailability. Finally, the rule *R* checks if the availability of a service is greater than *K*.

In the template, <*CaseId*> refers to the unique id of the certificate to be generated. It should be noted that EVEREST would receive primitive call and response events from the sensor that has been selected by SMaRT. Therefore, a call to the monitored service occurred at *t* is considered as served if a corresponding response occurs within a predefined time range between t and t+d. The value of *d* is denoted as <*D*> in the templates. During the translation process concrete values of <*_SrvId*>, <*CaseId*> and <*D*> are chosen according to a predefined set of criteria.

### D. Monitoring process & reaction to changes

During the monitoring state in the lifecycle model of Figure 2, EVEREST analyses the events received from sensors and detects violations of the monitoring rule that provides the concrete operational specification of the security property of the certification model. EVEREST attempts to match events sent to it from sensors with the monitoring rules and assumptions. When the *body* of a monitoring rule is fully matched (instantiated) with events, it will expect to receive events matching the rule's *head* or find them in its internal event database according to the designated time constraints. If such events are not found/received the rule is treated as violated. Otherwise, it is treated as satisfied. The same process is used for assumptions, except that when the *body* of an assumption becomes fully instantiated (i.e., True), the predicates in its *head* are derived and recorded in the EVEREST's database. Instances of monitoring rules instantiated by the events that match them constitute the monitoring results, which are sent to CG (see [15] for a detailed description).

In the case of the availability property example, EVEREST would report to CG instances of *R* where *sum(_P[]) / (t2 – _lmsTime) > K* (property satisfaction cases) as well as cases where *sum(_P[]) / (t2 – _lmsTime)* $\leqslant K$ (property violations). *_P[]* in these results will provide all the durations of unavailability periods and the events instantiating rule R will provide the concrete evidence demonstrating satisfaction or violation of the property.

 ISBN: 978-1-61208-298-1 

### E. Certificate provision

When a request for a certificate regarding the availability of a the service is made to CeRTiN, if the certificate type under monitoring is in the state *CanBeIssued*, CG will extract the monitoring results from its own database and generate the certificate. Prior to generating it however, it will also check if any extra validity tests required by the certification model are satisfied.

### F. Tool Support

An early proof-of-concept implementation of CeRTiN has been developed using two components of the open source monitoring framework developed by the SLA@SOI project, namely SMART and EVEREST. These two components play the roles of monitoring configuration selection tool and monitors in CeRTiN. Both of these tools have been implemented in Java based on Eclipse Modeling Framework (EMF). The tool is shown in Figure 7.

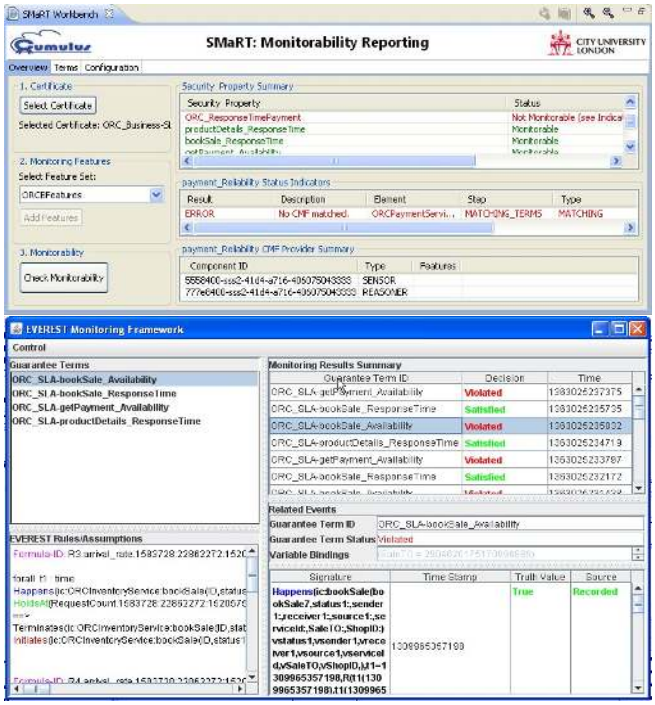


Figure 7: CeRTiN prototype

The upper part of Figure 7 illustrates SMART. As shown in the figure, the left hand panel of the tool allows the user to select the certification model to be applied for a given service and the monitoring features (i.e., capabilities) that are available in a cloud infrastructure.

The lower part of Figure 7 shows monitoring results produced by EVEREST. The left top panel of EVEREST lists the concrete security property that has been produced from the abstract security property that is to be monitored. The bottom left panel shows the monitoring specification of the abstract property (concrete security property) in EC–Assertion. The top right panel of the tool shows the summary of the monitoring results and bottom right panel shows the detail description of each monitoring result.

## V. RELATED WORK

Our approach is related to research strands in three different areas, namely: software and software services certification, cloud security and cloud monitoring.

Existing approaches in the field of security certification have focused on concrete software components and provide human readable certificates. Thus, they cannot support service-based scenarios that require machine-readable certificates and could support dynamic service selection and composition [11].

Unlike traditional approaches, the FP7 Project ASSERT4SOA [22] has been focusing on formal and test-based certification of services and has developed a framework for representing and using machine-readable service security certificates, known as ASSERTS, in service discovery and composition [21][22]. But overall research on the certification of cloud services and applications is still in an early stage. Some work has been done to predict the potential benefits of integrating certification schemes within cloud infrastructures [2][18], without, however, offering a concrete solution to this problem. Grobauer et al. [2], examine potential vulnerabilities in cloud computing, and acknowledge that the existence of such vulnerabilities is the lack of certification schemes and security metrics for cloud. Heiser and Nicolett [18] evaluate the cloud security risks and propose sharing IT risks with any externally provided service. A test based cloud certification approach has been proposed in [23] focusing on minimising test generation and execution activities in certifying cloud services. Test based certificates of cloud services could be combined with monitoring based certification, so as to produce extended hybrid certificates for the properties of interest.

More work has focused on auditing cloud security. The Cloud Controls Matrix (CCM) of the Cloud Security Alliance (CSA) [5], for example, contains a comprehensive set of controls to assess the information security assurance in clouds and maps controls to existing frameworks such as PCI DSS [30], COBIT [7]. CCM is currently being developed through the Open Certification Framework (OCM) [28] into a 3rd party certification program. Moreover, CSA has published the Cloud Audit protocol [4], which provides an automated query interface to cloud services for audit. Our approach is compliant with CSA's frameworks and we are investigating the potential integration of CeRTiN into OCM.

Cloud monitoring has been supported by several monitoring systems, including commercial software, open source systems, and research prototypes. Most of these systems (e.g., [25][27]) focus on performance monitoring without checking security properties as such. Cloud security monitoring is supported by positioning agents at different key points of a cloud infrastructure to detect incidents [13][14]. Other systems (e.g., DeSVi [32]) support the detection of SLA violations. None of these systems, however, supports security monitoring directly and customers can only use them to establish performance baselines, which, if deviated substantially, could indicate some incident conditions. EVEREST [15], the monitoring

framework that underpins CeRTiN, is in exception as it provides direct support for security monitoring.

## VI. CONCLUSION AND FUTURE WORK

In this paper, we have introduced a certification approach for the security of cloud services that is based on incremental assessment of security properties based on continuous monitoring, and have described the basic models and mechanisms for realising it.

The use of evidence coming from continuous monitoring provides coverage of contextual conditions that might not be possible to envisage, test or simulate through other forms of assessment of security properties (e.g., testing and static analysis) before deploying a cloud service. Therefore, our approach provides an advantage over test and static analysis approaches. We should note, however, that our work is part of a broader research programme undertaken by the EU F7 project CUMULUS, which aims to develop certification schemes that will enable the development of hybrid certification schemes, incorporating multiple types of evidence, including testing, static analysis and monitoring. Hence, we also aim to investigate how to integrate the certification models described in this paper with test and static analysis based models.

Furthermore, we are looking into the further development of some elements of our approach, notably the development of mechanisms for filtering low level monitoring data and aggregating them into compound forms of evidence prior to including them in certificates. This will be necessary in order to produce scalable and auditable certificates. We are also planning an evaluation activity based on inputs from certification authorities and industrial stakeholders that participate in CUMULUS.

## ACKNOWLEDGMENT

This work has been partially funded by the EU F7 project CUMULUS [8] (grant no 318580).

## REFERENCES

[1] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, “Security and Cloud Computing: InterCloud Identity Management Infrastructure”, In 19th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, 2010, pp. 263-265.

[2] B. Grobauer, T. Walloschek, and E. Stocker, "Understanding Cloud Computing Vulnerabilities," IEEE Security & Privacy, vol. 9, 2011, pp. 50-57.

[3] C. Cachin, I. Keider, and A. Shraer, “Trusting The Cloud”, IBM Research, Zurich Research laboratory, 2009.

[4] Cloud Security Alliance, Cloud Audit, Available from: https://cloudsecurityalliance.org/research/cloudaudit/

[5] Cloud Security Alliance, Cloud Controls Matrix, Available from: https://cloudsecurityalliance.org/research/ccm/

[6] Cloud Security Alliance, Security Guidance for Critical Areas of Focus in Cloud Computing vol. 2, available from: http://www.cloudsecurityalliance.org/guidance/csaguide.v2.1.pdf [retrieved: June, 2013].

[7] COBIT, http://www.isaca.org

[8] CUMULUS project, http://www.cumulus-project.eu/

[9] D. Catteddu and G. Hogben, “Cloud Computing: Benefits, Risks and Recommendations for Information Security.”, European Network and Information Security Agency (ENISA), 2009.

[10] D. S. Herrmann, “Using the Common Criteria for IT security evaluation”, Auerbach Publications, 2002.

[11] E. Damiani, and A. Mana, “Toward WS-Certificate”, In Proc. of the ACM Workshop on Secure Web Services (SWS 2009), 2009, pp. 1-2.

[12] F Lombardi and R Di Pietro, “Secure virtualization for cloud computing”, J. Netw. Comput. Appl. 34, 4, July 2011, pp. 1113-1122.

[13] F. Doelitzscher, C. Reich, M. H. Knahl, and N. L. Clarke, "Incident detection for cloud environments", Proc. of the Third International Conference on Emerging Network Intelligence,2011, pp. 100-105.

[14] F. Doelitzscher, C. Reich, M. Knahl, A. Passfall, and N. Clarke, "An agent based business aware incident detection system for cloud environments", Journal of Cloud Computing: Advances, Systems and Applications vol. 1:9, 2012.

[15] G. Spanoudakis, C. Kloukinas, and K. Mahbub, “The SERENITY Runtime Monitoring Framework”, In Security and Dependability for Ambient Intelligence, Springer, 2009, pp. 213-238.

[16] H. Foster and G. Spanoudakis, “Advanced Service Monitoring Configurations with SLA Decomposition and Selection”, In Proc. of 26th Annual ACM Symposium on Applied Computing, 2011, pp. 1582-1589.

[17] H. Li, Y. Dai, and B. Yang, "Identity-Based Cryptography for Cloud Security", IACR Cryptology ePrint Archive, 2011, pp.169-169.

[18] J. Heiser and M. Nicolett, “Assessing the Security Risks of Cloud Computing”, Gartner technical report, June 2008.

[19] K. Kearney, F. Torreli, and C. Kotsokalis, “SLA*: An Abstract Syntax for Service Level Agreements”, In Proc. Of 10th IEEE/ACM International Conference on Grid Computing, 2010, pp. 217-224.

[20] K. Mahbub, G. Spanoudakis, and T. Tsigkritis, "Translation of SLAs into Monitoring Specifications", In Service Level Agreements for Cloud Computing, R. Yahyapour, P. Weider (Eds), Springer-verlag, 2011.

[21] L. Pino, G. Spanoudakis, “Constructing Secure Service Compositions with patterns” 2012 IEEE 8th World Congress on Services, 2012, pp. 184-191.

[22] M. Anisetti et al., “ASSERT4SOA: Toward Security Certification of Service-Oriented Applications”, OTM 2010 Woekshops, 2010, pp. 38-40.

[23] M. Anisetti, C. A. Ardagna, and E. Damiani, “A Low-Cost Security Certification Scheme for Evolving Services”, In Proc. Of IEEE 19th International Conference on Web Services, 2012, pp. 122-129.

[24] M. Jensen, J. Schwenk, N. Gruschka, and L. L. Iacono, “On Technical Security Issues in Cloud Computing”, In Proc. of the IEEE International Conference on Cloud Computing, 2009, pp. 109-116.

[25] M. L. Massie, B. N. Chun , and D. E. Culler, “The ganglia distributed monitoring system: design, implementation, and experience”, Parallel Computing, vol. 30, 2004, pp. 817–840.

[26] Microsoft, The Economics of Cloud for the EU Public Sector, Paper, 2010:http://www.microsoft.eu/Portals/0/Document/EU_Public_Sector_Cloud_Economics_A4.pdf [retrieved: June, 2013].

[27] Nagios, 2011. http://www.nagios.org/

[28] Open Certification Framework, https://cloudsecurityalliance.org/research/ocf/

[29] Ovum, http://www.computing.co.uk/ctg/news/2113323/ovum-public-cloud-services-market-explode

[30] PCI Security Standards Council, PCI DSS Quick Reference Guide: Understanding the Payment Card Industry Data Security Standard v2, https://www.pcisecuritystandards.org/documents/PCI%20SSC%20Quick%20Reference%20Guide.pdf [retrieved: June, 2013]

[31] S. Kamara and K. Lauter, “Cryptographic cloud storage”, 14th International Conference on Financial cryptograpy and data security, 2010, pp. 136-149.

[32] V. C. Emeakaroha, R.N. Calheiros, M.A.S. Netto, I. Brandic, and C.A.F. De Rose, “DeSVi: An Architecture for Detecting SLA Violations in Cloud Computing Infrastructures” 2nd International ICST Conference on Cloud Computing, 2010.

 ISBN: 978-1-61208-298-1 