Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 7 (2012) pp.137-146
© MIR Labs, www.mirlabs.net/jias/index.html

A SLA-based interface for Secure Authentication Negotiation in Cloud

Massimiliano Rak[1], **Loredana Liccardo**[2] **and Rocco Aversa**[3]

[1] Dipartimento di Ingegneria dell'Informazione
Second University of Naples, Italy
{massimiliano.rak, loredana.liccardo, rocco.aversa}@unina2.it

Abstract: **Cloud Computing is a new computing paradigm. Among the incredible number of challenges in this field two of them are considered of great relevance: SLA management and Security management. A Service Level Agreement (SLA) is an agreement between a Service Provider and a Customer that aims at offering a simple and clear way to build up an agreement between the final users and the service provider in order to establish what is effectively granted in terms of quality. Cloud Computing assumes that everything from hardware to application layers are delegated to the network, accessed in a self-service way and following a pay-per-use business model. Security issues are related to the delegation to the network. The level of trust in such context is very hard to define and is strictly related to the problem of management of SLA in cloud applications and providers. In this paper we will try to show how it is possible, using a cloud-oriented API derived from the mOSAIC project, to build up an SLA-oriented cloud application which enables the management of security features related to user authentication and authorization to an Infrastructure as a Service (IaaS) Cloud Provider. As Cloud Provider we will adopt the PerfCloud solution, which uses GRID-based solutions for security management and service delivery. So the proposed solution can be used in order to build up easily a SLA-based interface for any GRID system.**

Keywords: SLA, Security, Cloud, Negotiation, GRID, Authentication

I. Introduction

Following the NIST definition, Cloud computing is *"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.[...]"*.
In recent years a lot of effort was spent both in academic and in business world exploring the effect of such paradigm and the way in which it can be applied. Among the incredible number of challenges in this field two of them are considered of great relevance: SLA management and Security management. In Cloud the services are out of customers direct control, i.e. a customer does not have a control of physical resources. For this reason, a customer, for the success of his business, is interested in obtaining: (i) the desired functionality (functional requirements), (ii) a certain availability, reliability, performance and security of resources that means a certain "Quality of Service" (non-functional requirements). This can be guaranteed by Service Level Agreement (SLA).
A *Service Level Agreement (SLA)* is an agreement between a Service Provider and a Customer, that describes the Service, documents Service Level Targets, and specifies the responsibilities of the Provider and the Customer. Service Level Agreements (SLAs) aim at offering a simple and clear way to build up an agreement between the final users and the service provider in order to establish *what is effectively granted* in terms of quality.
Regardless of importance of the SLA in Cloud Computing, there are few providers who provide SLA-support. The SLA management consists of several phases, i.e. negotiation, monitoring.
According to the NIST definition, one of the characteristics of Cloud paradigm is "on-demand self-service", that is "a consumer can provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service's provider". The main challenge is to realize an automatic SLA management that is automating the process of negotiation and monitoring.
As previously outlined, Cloud Computing assumes that everything from hardware to application layers are *delegated* to the network, accessed in a self-service way and following a pay-per-use business model; as a consequence Service level Agreements, enabling both users and provider to formalize what is offered, assume a great importance. In section VII we will show how many (of not all) of the most important cloud-related projects are considering SLA management as one of the key topic in their frameworks.
Security issues are related to the delegation to the network: is it possible for a final user to completely *trust* in a cloud provider, considering that everything is delegated to him? The *level* of trust in such context is very hard to define and is strictly related to the problem of management of SLA in cloud applications and providers.
There are a lot of different security aspects involved in such a context; The Cloud Security Alliance proposed fourteen different domain of security problems. In this paper we will focus only on the aspect of identity management, with focus on the mechanisms adopted to authenticate the users. Both

users and providers have strict requirements on the authentication mechanisms, because it affects the trade-off between simplicity of use, performance of access, management of the credentials and level of trust assured by a given security authentication mechanism.
In this paper we will try to show how it is possible, using a cloud-oriented API derived from the mOSAIC project [1, 2, 3], to build up an SLA-oriented cloud application which enables the management of security features related to user authentication and authorization to an Infrastructure as a Service (IaaS) Cloud Provider. As Cloud Provider we will adopt the PerfCloud solution [4], which uses GRID-based solutions (GSI and Globus [5]) for security management and service delivery. So the proposed solution can be used in order to build up easily a SLA-based interface for any GRID system.
The remainder of this paper is organized as follows, next section offers a simple and semi-formal description of SLA and security management in cloud providers using a simple and real case study. Section III depicts a reference scenario which describes the problem we want to solve and an analysis of the SLA-based Security management requirements. Section IV describes more in detail the case study, summarizing the technologies and frameworks adopted to solve the problem. Section V focuses on the solution proposed and its architecture while section VI outlines the requirements for an application which offers a SLA-based interface for security management. Finally, last two sections describe the related work, conclusions and future works.

II. SLA and Security: Problem Formalization

Building an agreement between users and provider on security features is a complex task, because it involves two opposite trade-offs, as shown in figure 1, both users and providers request that the partner respect some security features, moreover both offer some security grants. Each actor aims at offering as few grants as possible but will like to obtain as feature as possible.
As an example an user will like to have as much security features as possible from the provider, like the privacy on its data or high availability, but he will like to have an access as simple as possible, without complex authentication procedures. On the other side the provider will like to grant as less as possible (as an example do not grant that none access your data, so in case of intrusion they do not have to pat for penalties, or limited availability) but, on the other side will like to request as strong authentication procedure as possible in order to be protected against intrusions.
The role of Service Level Agreement is to offer a clear way to agree between Users and providers about what is offered and granted by each actor. As a consequence it is needed to build up a way to clarify the security features offered and granted by each partner involved in the agreement. As a matter of fact it is very hard to identify such templates in a general way, so the common solution is the development of custom templates for each new architecture and system.
Adoption of Service level Agreements (from now on SLA) for management of security features is an open problem in the cloud environment as previously outlined (moreover a deeper analysis of related work will be provided in section VII), in order to offer a clear approach and definition of this problem in this paper we will focus on a real case study: the building of an SLA-based interface for an IaaS Cloud provider.
The provider is built using the PerfCloud [4] framework, which is based on the integration of GRID and Cloud, following the cloudgrid [6] approach. It is out of the scope a detailed analysis of such solution, but it is interesting to point out that it uses the GSI framework for management of user authentication and authorization, this implies that the solution proposed can be used for any GRID system.

III. SLA-based Security management: Our Reference Scenario

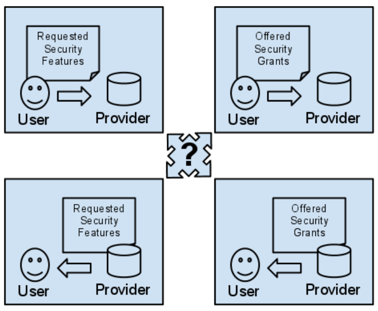
In this section, we present a reference scenario, depicted in figure 2, which is the basis for describing the problem (discussed in the previous section) and that we want to solve. We will provide an analysis of the SLA-based Security management requirements, tackling the problem. As shown, we consider a Cloud Provider that offers an Infrastructure as a Service (IaaS). In particular, the provider offers different set of services (from 1 to 5 in picture), applying different security policies. Some service set are offered adopting several policies (in the picture the service set 1) others are offered only following a given policy. This means that the provider can offer:

- different services, accessible with different security levels.
- a same service, accessible with different security levels.

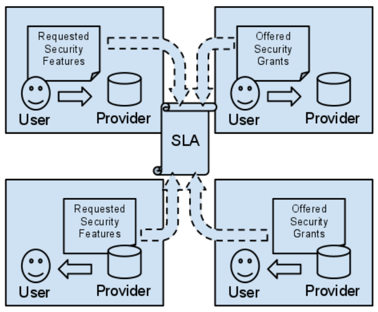
Suppose a user wants access to a Cloud service. The user, after the registration with the cloud provider, obtains a set of credentials, which he will use to invoke a service. Specifically, consider a user wishes access to a certain set of services (in picture service Set X).
To solve the problem, the Cloud Provider must employ a *SLA management system*. This system will allow:

- a user to negotiate, with the cloud provider, the service and the relative security level. Once an agreement between a user and a cloud provider is reached, we can define a negotiation's outcome as a SLA agreement.
- a cloud provider to enforce a SLA agreement.
- to manage a SLA, keeping track of its status; for example, by exploiting this functionality, a user can check the status of his SLA request, while the provider can easily detect the SLA active.

To realize this system, the Cloud Provider offers a set of SLA templates which contains a description of the service set offered, the credentials requested to the user and the policy applied. In order to manage the SLA-based access to services, the final user submits to the system the SLA template of his interest, and the Cloud provider, once checked its applicability, can reach or no a SLA agreement with this user, concluding the negotiation. In fact, this is a simple negotiation based on a single round ”offer, accept” the according the WS-Agreement protocol, which will be outlined in section



(a) The Matching problem



(b) The role of SLA

Figure. 1: The security agreement problem and the role of SLA

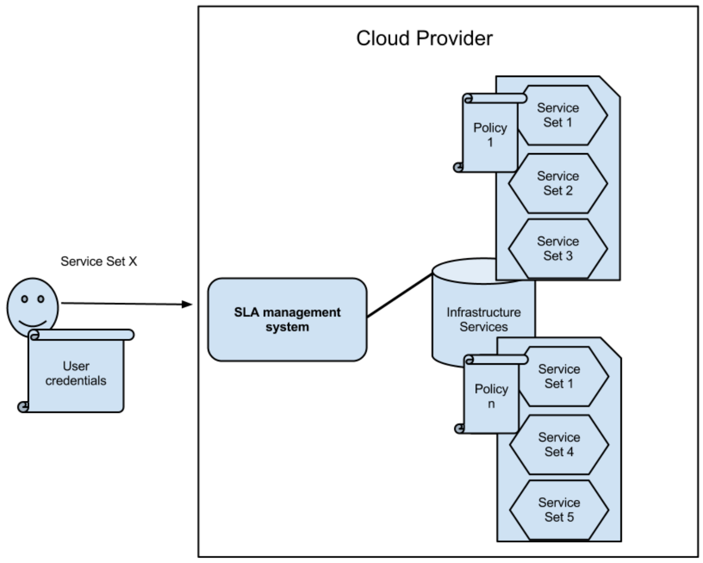


Figure. 2: SLA-based interface for security management in cloud environment

IV. Then, the cloud provider, to perform the enforcement of a SLA agreement, exploits an authorization system to enable/disable access of a user to cloud service sets. Once a user negotiation is terminated, the provider configures as a consequence this authorization system in order to accept such a user. Users are granted that the services (and resources) accessed are protected by the provider following a known policy. Providers are granted that users have the right credentials (the services are protected by the given policy) and are able to audit the agreement acquired, knowing when each user has requested a more secure service.
In section V and VI we propose an architectural solution to solve this problem. In particular we present a SLA management solution which is Cloud Provider independent.
Moreover, knowing that more secure services often have greater costs in term of performance and resource consumption, it is possible to manage a trade-off between the resource usage, the security offered (and eventually the global cost of the service).

IV. The Case Study: SLA interface towards PerfCloud

In section III we described the problem of building an SLA interface towards an IaaS Cloud provider focusing on a reference scenario. In the following, we provide a detailed description of the approach used to address such problem in the context of a given specific solution: the PerfCloud framework on which we build an IaaS Cloud Provider. The solution we propose founds on the adoption of two main frameworks: GSI, that is the security module in Globus and many other GRID toolkits, and mOSAIC, which is a framework for Cloud application development. Moreover, we adopt a WS-Agreement specification to build up a SLA management system. In the following subsections we briefly summarize the main concepts of such solutions.

A. PerfCloud: the cloudgrid approach

As Cloud Provider we adopt PerfCloud, which is based on the integration of GRID and Cloud, following the cloudgrid approach. It is out of the scope of this paper a detailed description of such solution which can be found in [4, 6].
The basic configuration of PerfCloud founds on Cloud on GRID approach: using a GRID middleware (Globus GT4) it offers cloud oriented services, such as delivery and management of virtual machines (start, stop, sleep, ..).
This approach is shared with other common projects, like Nimbus [7].
The Globus middleware offers the basis for building a Service oriented interface and the components for configuring and manage the security of the system in terms of authentication and authorization (through GSI as will be outlined in in the following).
In [8] the authors outlines how management of cloud services implies the needing of ad-hoc consideration related to the security configuration, due to the presence of new actors (Cloud and GRID users must have different access rights, as an example). Such actors implies the needing of adoption of authentication and authorization mechanisms able to apply complex policies, which are clearly proposed in [8]. Adoption of such technical solution (i.e. an RBAC-style authorization mechanisms like XACML or Shibboleth) can be introduced in a GRID environment and have an acceptable trade-off, as shown in [9].
The actors identified and their respective roles (and so rights of access) can be summarized as follows:

- **System Administrator**: can manage the physical machines from hardware up to the operating system level. He is responsible for installing, configuring and starting the GRID platform and its Certification Authority, for managing GRID identities and accounts, for updating the security policies on the system;
- **Grid User**: can create and use GRID resources;
- **Cloud Administrator**: is a GRID User with additional rights. He can supervise the cloud environment creating/maintaining new Virtual Clusters and managing Cloud User rights. In particular, he can enable/disable a Cloud User for the access to one or several Virtual Clusters;
- **Cloud User**: is a GRID user with additional rights. He can turn on/off, access, use, configure Virtual Clusters previously assigned to him by the Cloud Administrator.

Moreover, PerfCloud architecture enable us to configure the globus container in order to restrict the access to users not only on the basis of their credentials (i.e. if they have or not a valid certificate) and respect to their role (which is defined using the XACML authorization engine), but even in the way in which he effectively secures the message exchange with the cloud provider (as an example without cryptography, using SSL or using application level cryptography like `SecureMessage` or `SecureConversation`. This last configuration enable to offer different security levels, even if with an higher cost (as outlined in [9]).
Following the problem model outlined in section II we can assume the following needings for both the side of the SLA agreement:

- **Provider** have a different set of services for each user role (i.e. a service set for Cloud administrators, a service set for Cloud Users, . . .). Each service set is protected with different policies of access. Each service set has an associated Service Level Agreement template which describe the credentials and authentication roles needed by each different user.
- **User** must have a valid certificate in order to access the PerfCloud environment. When the user need SLA-protected services he must negotiate with the SLA system his role in the system and the security level he need.

It is important to point out that more restrictive security policies (as an example adoption of Transport security layer) offer grants to final users about the confidentiality of access to his resources (the virtual machines) and offer grants of correct usage to the system provider.

B. Security configuration in GSI

Globus (GT4) and Globus security Infrastructure (GSI) offer a flexible mechanism to enforce authentication and authorization. This is based on the adoption of security description

files that describe both authentication requirements, along with authorization policy decision points where role-based access control policies are evaluated [10]. The authorization decision can be performed by standard GSI components or by external authorization service as XACML [11], PerMIS [12] or gridShib [13].
In particular, GT4 uses the concept of security descriptors as standard method for configuring the security requirements and policies of clients and services. GT4 provides four different type of security descriptors:

- **Container security descriptor** specifies the container level security requirements that need to be enforced, i.e., the authentication and authorization mechanisms adopted to let a user access services and resources in the container;
- **Service security descriptor** specifies the service level security requirements that need to be enforced, i.e., the authentication and authorization mechanisms adopted to let a user access a given service;
- **Resource security descriptor** specifies the resource level security requirements that need to be enforced, i.e., the authentication and authorization mechanisms adopted to let a user access a given resource;
- **Client security descriptor** are adopted into the GT4 clients to specify the security mechanisms to be adopted on service invocations.

Container, service and resource security descriptors have different priority levels. The most restrictive policy is applied at the resource level, and overrides the others. Service and container security descriptors are configured as XML files, defined in the deployment descriptor and locally stored. On the other hand, resource security descriptors can only be dynamically created, either programmatically or from a descriptor file. GT4 offers APIs to define the security descriptor into the client code or as an XML file.
The typical structure of an XML file containing the configuration of a security descriptor of container, service or resource type, contains two main elements: *auth-method* and *authz*, as in the following:

Listing 1: Example of XML based Security Descriptor

```
<securityConfig xmlns="http://www.globus.org">
      <auth-method>
         <GSISecureConversation>
            <protection-level>
               <integrity/>
            </protection-level>
         </GSISecureConversation>
      </auth-method>
   </method>

   <authz value="pdp1:org.foo.PDP1 pdp2:org.
       foo.PDP2
       foo1:org.foo.authzMechanism bar1:org.
           bar.barMechanism"/>

</securityConfig>
```

As for authentication, GT4 uses digital certificates to authenticate and delegate users. Furthermore, GSI allows to enable security at transport level and at message level. Transport-level security means that the complete communication (all the information exchanged between a client and a server) is encrypted. With message-level security, only the contents of the SOAP message are encrypted. GSI offers two message-level protection schemes (*GSISecureMessage* and *GSISecureConversation*), and one transport-level scheme (*GSITransport*).
Summarizing, four types of authentication methods are provided by GT4:

- **none**: no authentication is performed.
- ***GSISecureMessage***: each individual message is encrypted.
- ***GSISecureConversation***: a secure context is first established between client and server; all the following messages can reuse that context.
- ***GSITransport***: transport-level security is provided by using TLS.

In addition, with *GSISecureMessage*, *GSISecureConversation* and *GSITransport*, the security administrator can specify the protection levels *integrity* (data are signed) and *privacy* (data are encrypted and signed). Clients must be configured to adopt a compliant authentication mechanism.
As regards authorization, container, services and resources can also be protected by different authorization mechanisms (enforcing different Policy Decision Points - PDP) with different mechanisms for collecting attributes (Policy Information Points - PIP).

C. Programming the Cloud with mOSAIC

The full mOSAIC solution includes four main modules: mOSAIC API, mOSAIC Framework or platform and the mOSAIC provisioning system and the semantic engine, in this section we will focus on the relationship between API, Framework and Provisioning system.
The role of the provisioning system is to decouple the cloud applications from the cloud providers: mOSAIC Developers never focus on Cloud Provider specific resources, but they will refer to resources trough abstraction which offer uniform access to them, independently from the provider and the technologies they support. The provisioning system works mainly at Infrastructure as a Service (IaaS) level, managing resources like virtual machine, cloud storages, communication systems. The role of the Provisioning System in mOSAIC is assigned to the Cloud Agency [14].
The mOSAIC Framework is a collection of cloud components which can be run independently and offer a clear set of services. The mOSAIC predefined components aim at offering simple and common services which can be composed easily by mOSAIC developers in order to build up complex applications. Example of mOSAIC components are HTTPgw (HTTP gateway) a component which offer an HTTP interface and forwards messages on cloud queues. The mOSAIC framework defines a very simple Platform as a Service solution which enables the execution of a complex services with well known interfaces. The ratio of the Framework is to offer a simple way to reuse a large set of existing technologies and solution when building up a custom application.

Due to the existence of mOSAIC Provisioning System (Cloud Agency) and of the mOSAIC Framework, the role of the API can be now well outlined: they should offer a programming model and its implementation in a given programming language (java and in the future python) in order to build up applications which consume cloud resources and easily interact with a large set of different technologies. It is important to put in evidence that mOSAIC API are not wrappers which enable transparent access to cloud provider for well known and predefined set of resources (like other solution like jclouds or Apache Delta Cloud solutions does) but a completely different way of thinking cloud application, in which the resources are modeled just in terms of the functionalities offered, not on the way in which they are accessed and in which they internally work. As an example for a mOSAIC application a cloud storage can be a Key Value store system, but it never cares if after this definition there are an Amazon S3 instance or a cluster of virtual machines running Riak. mOSAIC Developers have access to a new set of concepts (Cloudlet, Connector) which help in thinking cloud applications in a way which focuses on cloud resources and cloud communications and, as a consequence, exploit at best the scalability, elasticity, self-adaptiveness features which Cloud paradigm offers. Problems related to quality management of mOSAIC application are managed as a consequence, through a dedicated Service Level Agreement (SLA) based model and mOSAIC components devoted to quality management.

D. mOSAIC API Concepts

In mOSAIC a Cloud Application is developed as a composition of inter-connected building blocks. A Cloud ”Building Block” is any identifiable entity inside the cloud environment. It can be the abstraction of a cloud service or of a software component. It is controlled by user, configurable, exhibiting a well defined behavior, implementing functionalities and exposing them to other application components, and whose instances run in a cloud environment consuming cloud resources.
Simple examples of components are: a Java application runnable in a platform as a service environment; or a virtual machine, configured with its own operating system, its web server, its application server and a configured and customized e-commerce application on it. Components can be developed following any programming language, paradigm or *in-process* API. An instance of a cloud component is, in a cloud environment, what an instance of an object represents in an object oriented application.
Communication between cloud components takes place through cloud resources (like message queues – AMPQ, or Amazon SQS) or through non-cloud resources (like socket-based applications).
Cloudlets are the way offered to developers by mOSAIC API to create components. Cloudlet runs in a cloudlet container which is managed by the mOSAIC Software platform. A Cloudlet can have multiple instances, but it is impossible at runtime to distinguish between the cloudlet instances. When a message is directed to a cloudlet it can be processed by anyone of the cloudlet instances. The number of instances is under control of the cloudlet container and is managed in order to grant scalability (respect to the cloudlet workload). Cloudlet instances are stateless, Cloudlets use cloud resources through connectors. Connectors are an abstraction of the access model of cloud resources (of any kind) and are technology independent. Connectors control the cloud resource through technology-dependent *Drivers*. As an example, a cloudlet is able to access to Key-Value store systems through a KVstore Connector, it uses an interoperability layer in order to control a Riak, or a MemBase KV driver. Further details on the mOSAIC programming model, which is out of the scope of this paper can be found in [2], [15], [1].

E. Security Parameters for WS-Agreement

Our SLA management solution adopts a WS-Agreement specification, a Web Services protocol for establishing an agreement between two parties, such as between a service provider and consumer. The objective of the WS-Agreement is to define an XML language and a protocol for advertising the capabilities of service providers and creating agreements based on creational offers. This specification provides a schema for defining overall structure for an agreement document and a template. The goal of WS-Agreement is to standardize the terminology, concepts, overall agreement and template structure. Moreover, the WS-Agreement standard defines a state model for an agreement, because during a SLA management, a SLA can assume several states (This state model will be treated in section VI).
The creation of an agreement can be initiated by the service consumer side or by the service provider side and typically starts with a pre-defined agreement template. In particular, the overall process starts (optionally) with the initiator retrieving a template from the responder, continues with the initiator making an offer and is concluded when the responder accepts or refuses the offer. In fact, the WS-Agreement protocol is based on a single round ”offer, accept” message exchange: the initiator sends an obligating offer, through a template, which the responder may accept or reject. This is a simple negotiation. Furthermore, an agreement template is an XML document used by the agreement responder to advertise the types of offers it is willing to accept [16].
One of the open problem in building such solution is the missing of a standard for management of the security parameters to be negotiated with final users. WS-Agreement offers a general purpose container, which is completely parameter-independent. In order to build up this SLA application, we had to build up a new set of parameters to be inserted in WS-Agreement. These custom parameters can be used for any GRID environment which adopts the GSI module (i.e. both globus and gLite).

Listing 2: Security Description

```
<ws:Context>
    <wsag:AgreementInitiator> /O=Grid/OU=
        GlobusTest/OU=simpleCA-pc/CN=bacon </
        wsag:AgreementInitiator>
</ws:Context>

<ws:Terms>
    <ws:All>
      <ws:ServiceDescriptionTerm ws:Name="
          perfCloudVbox" ws:ServiceName="Vbox
          Service in PerfCloud">
        <sec:GSIAuth>
          <transport>HTTP</transport>
          <WSAuth>SecureConversation</WSAuth>
          <WSAuthZ>XACML</WSAuthZ>
        </sec:GSIAuth>
      </ws:ServiceDescriptionTerm>
    </ws:All>


     <wsag:GuaranteeTerm wsag:Name="
         SecurityLevel">
        <wsag:KPITarget>
            <wsag:KPIName>SecurityLevel</
                wsag:KPIName>
            <wsag:CustomServiceLevel>(
                Transport eq HTTP)AND(WSAuth
                eq SecureConversation)AND(
                WSAuthZ eq XACML)</
                wsag:CustomServiceLevel>
        </wsag:KPITarget>
    </wsag:GuaranteeTerm>
</ws:Terms>
```

As shown in listing 2 these parameters of WS-Agreement presented in the template (a document used by the agreement responder to advertise the types of offers it is willing to accept) are: (1) element Context that contains information about agreement parties. One of the elements presented in Context is AgreementInitiator. Its value is the DN (distinguished name) of the user that used services Globus. (2) element Terms that defines the content of an agreement. It contains element ServiceDescriptionTerm and element GuaranteeTerm. ServiceDescriptionTerm encloses a description of a service. We added element GSIAuth in which the user can specify parameters of security. GuaranteeTerm defines the assurance on service quality (or availability) associated with the service described by the service definition terms. It contains element KPITarget that defines service level objective (represents the quality of service aspect of the agreement) as an expression of a target of a key performance indicator associated with the service. The value of element CustomServiceLevel is a specific security level.

V. SLA-based Security management: Architecture

The previous section (sect. IV) presented a detailed description of a case study while section III described the problem. In this section we will focus on the solution proposed and its architecture. The global architecture of the solution is summarized in figure 3. The key choice of the solution proposed in this paper is to build up a SLA management system as an application using a cloud-based API (mOSAIC) and using the cloud resources offered in order to run it. We built up our application as a mOSAIC Application, modeled as a collection of components that are able to communicate each other through messages. Our mOSAIC application consists of several components; each component has a different well defined role. In the next section, it will be described the architecture of our SLA management application in detail. As shown in figure 3, our mOSAIC application is based on components and cloud resources which run on a virtual machine hosting the mOS operating system, a lightweight linux distribution.This application directly communicates with the Cloud Provider.
As regards the Cloud Provider, in order to manage the different roles of the users and the different security policies to be maintained we organized the PerfCloud configuration in three different containers, each of them hosted in a different virtual machine, running on the cluster Frontend and having their own static IP address. The three different containers are protected through different security policies and host a different set of services.
In order to manage the different security roles we organized the PerfCloud services in two main set: Cloud users and Cloud Administration sets. Services for common GRID users are, for now, out of the interests of this paper and may be offered on the same containers or on additional ones. Cloud Administrator Services, i.e. the ones that enable the creation of a completely new machine and/or virtual cluster and enable an user to use it, are offered only in the third container, whose policy is the most restrictive ones (both Transport and Message Security). Cloud User services (the ones which enable a user to start/stop a virtual cluster) are offered in two different containers, configured to have no cryptography on request messages (i.e. the lower level of security granted by just GRID proxy certificate) or using Secure message (the one which crypt each SOAP message). The first container offers less grants to the final user, but has a minimal overhead, the second one offers higher grants in terms of confidentiality, but has higher overhead (as shown in [9]). Details on such configuration can be found in the already cited papers.
The approach proposed is simple: once the user has obtained an agreement with the SLA management system, he will be authorized to access to one or more containers, following the agreed security requirements. The application which enables SLA management is built in order to allow a user to negotiate security features related to user authentication with cloud provider through a WS-Agreement template. When an agreement between a user and a cloud provider is reached, it contains a description of the security features and so the authorization file of the right container is configured as a consequence by a Cloud Provider. To reach this goal, on the cluster Frontend runs an application which communicates with the SLA management system. In particular this system, once an agreement is accepted, sends to the cluster Frontend some informations so then the authorization file of the right container is configured, in order to enable/disable access of users to the PerfCloud service sets.

VI. SLA Management Application

In this section we outline the requirements for an application which offers a SLA-based interface for management of security parameters. Such an application has the following main requirements:

- the SLA application should support the negotiation of security parameters through an agreement template.
- the SLA application should be able to acquire the template offered by the user and assume a decision about it (it is acceptable or not);
- the SLA application should be able to communicate with the Cloud Provider so the provider can configure the authorization engine following the SLA agreed with the users.

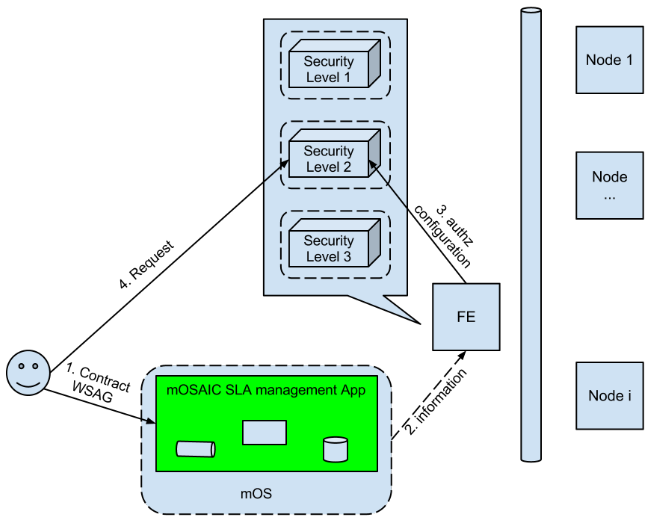


Figure. 3: System Architecture

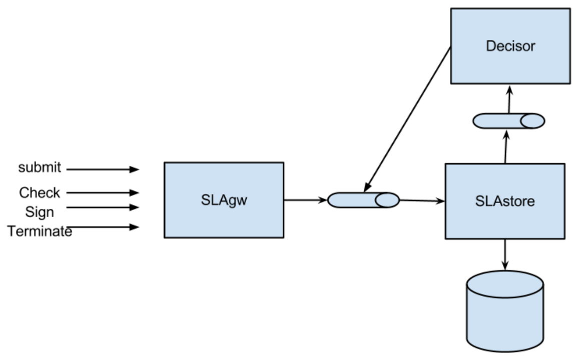


Figure. 4: SLA Application for security management

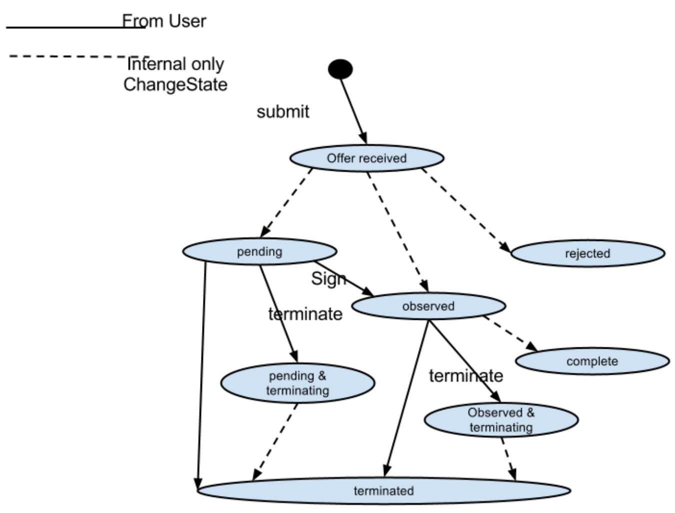


Figure. 5: WS-Agreement states

- the SLA application should be able to take trace of the SLA agreed.

The architecture is depicted in figure 4. As shown, our S-LA management application consists of several components and resources (offered by framework mOSAIC) as queue and storage, following the architecture proposed in [3]. It is a component-based application, in which each component realizes a well-defined functionality. Logically, the architecture consists of several core parts:

- `SLAgw` and `SLAstore`, which provides SLA management.
- `Decisor`, which evaluates if the SLA is acceptable.

Communication between components will take place through messages. In the following we propose a brief description of the behavior of each component and how they interoperate.
The *SLAgw* is the component which has both the role of offering the application as a Web application and implements a REST-based interface. The SLAgw REST API, accepts essentially the following methods:

- `submit`, with which a Customer submits the agreement or template to Provider.
- `check`, with which a Customer requests the status of SLA agreement.
- `sign`, with which a Customer accepts the agreement.
- `terminate`, with which a Customer requests the termination of SLA agreement.

These methods represent the mandatory operations for SLA management protocols.
SLAstore is the component which stores an agreement or template. It is a component that manages features such as: the capability of taking trace of the signed SLAs and the status of all the exchanged SLAs. In fact, as anticipated, during a SLA management, a SLA can assume several states, defined by WS-Agreement standard.
In particular, the figure 5 puts in evidence the WS-Agreement states. With a labeled solid line we indicate a change of state caused by a user through messages as: submit, sign, terminate. With a dashed line we indicate a internal change of state due to a system.
Decision is the component which evaluates if the SLA is acceptable according to the security features offered by a Cloud Provider. If an agreement is reached, then this cloudlet sends a message through a queue to the Cloud Provider. In fact, on the Cloud Provider runs a program which reads a message from a queue. The message contains security parameters with which the configuration file in the right container is updated.
The application behavior can be briefly summarized on the base of the messages exchanged by the components which have been described above and connected as in Figure 4. The *SLAgw* receives the messages from the final users through its SLA-oriented API. All the negotiation messages are sent through the *submit* REST call, except the ones needed to effectively sign the SLA. By a web interface the user can interactively send request and check responses.
Each message submitted by the SLAgw is intercepted by the *SLAstore* component. A message, received by the *SLAstore*, is an agreement or a template. SLAstore stores it in a local storage along with an associated information and relative to the state defined according to the standard WS-Agreement. Then the component forwards this message to *Decisor*. Once received the message, Decisor evaluates if the agreement or template is acceptable according to the security features offered by a Cloud Provider. In both cases (that is accepted or rejected), its decision causes a change of the WS-Agreement state. It communicates its decision to the SLAstore which updates the state information relative to the agreement or template. If an agreement is reached, then Decision sends a message to the Cloud Provider. This message contains security informations with which the Cloud Provider configures the authorization system.

VII. Related Work

To the best of our knowledge not much work has been done in the area of configuring security requirements specified through WS-Agreement documents.
Karjoth et al. [17] introduce the concept of Service-Oriented Assurance (SOAS). SOAS is a new paradigm defining security as an integral part of service-oriented architectures. It provides a framework in which services articulate their offered security assurances as well as assess the security of their sub-services. Products and services with well-specified and verifiable assurances provide guarantees about their security properties. SOAS enables discovery of sub-services with the right level of security. SOAS adds security providing assurances (an assurance is a statement about the properties of a component or service) as part of the SLA negotiation process.
Smith et al. [18] present a WS-Agreement approach for a fine grained security configuration mechanism to allow an optimization of application performance based on specific security requirements. They present an approach to optimise Grid application performance by tuning service and job security settings based on user supplied WS-Agreement specification. WS-Agreement describes security requirements and capabilities in addition to the traditional WS-Negotiation attributes such as computational needs, quality-of-service (QoS) and pricing.
Brandic et al. [19] present advanced QoS methods for meta-negotiations and SLA-mappings in Grid workflows. They approach the gap between existing QoS methods and Grid workflows by proposing an architecture for Grid workflow management with components for meta-negotiations and SLA-mappings. Meta-negotiations are defined by means of a document where each participant may express, for example, the pre-requisites to be satisfied for a negotiation, the supported negotiation protocols and document languages for the specification of SLAs. In the pre-requisites there is the element `security` that specifies the authentication and authorization mechanisms that the party wants to apply before starting the negotiation. With SLA-mappings, they eliminate semantic inconsistencies between consumer's and provider's SLA template. They present an architecture for the management of meta-negotiation documents and SLA-mappings and incorporate that architecture into a Grid workflow manage-

ment tool.
Chaves et al. [20] explore Service Level Agreements for Security or just Sec-SLAs, a formal negotiated document that defines in, specially, a quantitative way what service levels will be delivered from the provider to the customer. In particular, a Sec-SLA is a specific SLA that deals with metrics related to security instead of the traditional telecommunication metrics such as throughput, delay, packet loss and other similar metrics. The authors discuss some aspects of defining security metrics, specifically, the difficulties faced during the security metrics definition process and the Sec-SLA monitoring, as well as an analysis on the Sec-SLA role in new paradigms like cloud computing. In particular, an approach to be used when specifying Sec-SLAs for cloud computing and some security metrics suitable for cloud computing environments are presented.
Demchenko et al. [21] discuss a Cloud security infrastructure which provides consistent security services in on-demand provisioned Cloud infrastructure services. The paper refers to the architectural framework for on-demand infrastructure services provisioning, being developed by authors, that provides a basis for defining the proposed Cloud Security Infrastructure. In this work, the authors propose an SLA management solution build using WSAG4J framework, an implementation of WS-Agreement specification, to maintain important SLA guarantees during the provisioned services and relative to security parameters too.

VIII. Conclusions and Future Works

In this paper we have shown how it is possible to realize a SLA-based interface towards a Cloud provider. Specifically, using a cloud-oriented API derived from the mOSAIC project, it is possible to build up an SLA-oriented cloud application which enables the management of security features related to user authentication and authorization to an Infrastructure as a Service (IaaS) Cloud Provider. The application which enables SLA management is built in order to allow an user a simple negotiation of security features offered by a cloud provider. Once the user has obtained an agreement with the SLA management system, he will be authorized to access to one or more service set, following the agreed security requirements.
In the future, we will extend our work adding the mechanism of digital signature on a SLA agreement, offering to an user the possibility of signing his agreement. Moreover, using SLA, we will add parameters on performance to offer information on trade-off between security offered and guaranteed performance.

Acknowledgments

This research is partially supported by FP7-ICT-2009-5-256910 (mOSAIC).

References

[1] mOSAIC Project, "mosaic: Open source api and platform for multiple clouds," http://www.mosaic-cloud.eu, 2010.

[2] I. I. M. van Sinderen F. Leymann, B. S. S. – Science, and T. Publications, Eds., *Towards a cross platform Cloud API. Components for Cloud Federation*, 2011.

[3] IEEE, Ed., *User Centric Service Level Management in mOSAIC Application*, 2011.

[4] V. Casola, M. Rak, and U. Villano, "Perfcloud: Performance-oriented integration of cloud and grid," in *Cloud Computing*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, E. Dekel, O. Akan, P. Bellavista, J. Cao, F. Dressler, D. Ferrari, M. Gerla, H. Kobayashi, S. Palazzo, S. Sahni, X. S. Shen, M. Stan, J. Xiaohua, A. Zomaya, and G. Coulson, Eds. Springer Berlin Heidelberg, 2010, vol. 34, pp. 93–102.

[5] The Globus Security Team, "Globus toolkit version 4 grid security infrastructure: A standards perspective," 2005, www.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf .

[6] V. Casola, A. Cuomo, M. Rak, and U. Villano, "The cloudgrid approach: Security analysis and performance evaluation," *Future Generation Computer Systems*, no. 0, pp. –, 2011.

[7] University of Chicago, "Nimbus project," 2009, http://workspace.globus.org/clouds/nimbus.html.

[8] V. Casola, R. Lettiero, M. Rak, and U. Villano, "Access control in cloud-on-grid systems: The PerfCloud case study," in *Computers, Privacy and Data Protection: an Element of Choice*, S. Gutwirth, Y. Poullet, P. De Hert, and R. Leenes, Eds. Springer Netherlands, 2011, pp. 427–444.

[9] V. Casola, A. Cuomo, M. Rak, and U. Villano, "Security and performance trade-off in perfcloud," in *Euro-Par Workshops*, ser. Lecture Notes in Computer Science, M. R. Guarracino, F. Vivien, J. L. Träff, M. Cannatoro, M. Danelutto, A. Hast, F. Perla, A. Knüpfer, B. D. Martino, and M. Alexander, Eds., vol. 6586. Springer, 2010, pp. 633–640.

[10] D. Ferraiolo and D.R.Kuhn, "Role-based access control," in *Proc. of the 15th National Computer Security Conference*, 1992, pp. 554–563.

[11] The OASIS technical commitee, "Xacml: extensible access control markup language," 2005, http://www.oasisopen.org/committees/xacml/repository/.

[12] D. W. Chadwick, G. Zhao, S. Otenko, R. Laborde, L. Su, and T.-A. Nguyen, "Permis: a modular authorization infrastructure," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 11, pp. 1341–1357, 2008.

[13] T. Barton, J. Basney, T. Freeman, T. Scavo, F. Siebenlist, V. Welch, R. Ananthakrishnan, B. Baker, M. Goode, and K. Keahey, "Identity federation and attribute-based authorization through the globus toolkit, shibboleth, gridshib, and myproxy," in *Proc. of 5th Annual PKI R&D Workshop*, 2006.

[14] R. Aversa, B. D. Martino, S. Venticinque, and D. Petcu, “Agent based cloud provisioning and management, design and prototypal implementation,” in *1st International Conference on Cloud Computing and Services Science (CLOSER2011)*, I. I. M. van Sinderen Frank Leymann and B. Shishkov, Eds. ScitePress, 2011, pp. 184–191.

[15] IEEE, Ed., *Building an Interoperability API for Sky Computing*, 2011.

[16] The Open Grid Forum, “Web services agreement specification,” 2007, http://ogf.org/documents/GFD.192.pdf.

[17] G. Karjoth, B. Pfitzmann, M. Schunter, and M. Waidner, “Service-oriented assurance, comprehensive security by explicit assurances,” in *Quality of Protection*, ser. Advances in Information Security, D. Gollmann, F. Massacci, and A. Yautsiukhin, Eds., vol. 23. Springer US, 2006, pp. 13–24. [Online]. Available: $http : //dx.doi.org/10.1007/978 - 0 - 387 - 36584 - 8_2$

[18] M. Smith, M. Schmidt, N. Fallenbeck, C. Schridde, and B. Freisleben, “Optimising security configurations with service level agreements.”

[19] I. Brandic, D. Music, S. Dustdar, S. Venugopal, and R. Buyya, “Advanced qos methods for grid workflows based on meta-negotiations and sla-mappings,” *2008 Third Workshop on Workflows in Support of LargeScale Science*, 2008.

[20] S. A. de Chaves, C. B. Westphall, and F. R. Lamin, “Sla perspective in security management for cloud computing,” in *Proceedings of the 2010 Sixth International Conference on Networking and Services*, ser. ICNS ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 212–217. [Online]. Available: http://dx.doi.org/10.1109/ICNS.2010.36

[21] Y. Demchenko, C. Ngo, C. de Laat, T. W. Wlodarczyk, C. Rong, and W. Ziegler, “Security infrastructure for on-demand provisioned cloud infrastructure services,” in *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 255–263. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2011.42

Author Biographies

Massimiliano Rak is currently an Assistant Professor at the Information Engineering Department of the Second University of Naples, Italy. He received the MSc degree in Computer Science Engineering from the University of of Napoli Federico II in 1999. He got a Ph.D. in Computer Engineering from the Second University of Napoli in 2002. His research activities include both theoretical and experimental issues, in the areas of performance evaluation of computing systems, parallel and distributed software engineering, security of information systems, and they are documented by many publications, in national and international journals and conference proceedings.

Loredana Liccardo is a second year PhD student in Computer Engineering at the Information Engineering Department of the Second University of Naples, Italy. She received the MSc degree in Computer Science Engineering from the Second University of Naples in 2010. Her research interests include Cloud Computing, SLA and securiy management in cloud environment.

Rocco Aversa graduated in Electronic Engineering at University of Naples in 1989 and received his Ph.D. in Computer Science in 1994. He is Associate Professor in Computer Science at the Department of Information Engineering of the Second University of Naples. His research interests are in the area of parallel and distributed systems. The research themes include: the use of the mobile agents paradigm in the distributed computing; the design of simulation tools for performance analysis of parallel applications running on heterogeneous computing architectures; the project and the development of innovative middleware software to enhance the Grid and Cloud computing platforms.