DOI 10.12694/scpe.v14i4.934
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 4, pp. 291–306. http://www.scpe.org

DELEGATION ACROSS STORAGE CLOUDS: ON-BOARDING FEDERATION AS A CASE STUDY

CIRO FORMISANO[3] AND ELLIOT K. KOLODNER[2] AND ALEXANDRA SHULMAN-PELEG[2] AND ERMANNO TRAVAGLINO[3] AND GIL VERNIK[2] AND MASSIMO VILLARI[1] *

Abstract. As the volume of digital data rapidly increases, storage clouds are becoming a popular solution for both enterprise and personal data, and the number of storage cloud solutions is also increasing. However, these solutions do not yet deal with the need of customers for interoperability and data migration from one cloud to another. These issues can be addressed through federation of cloud infrastructures. An important aspect of federation is delegation of access control, where one actor, e.g., an end user, authorizes another actor, e.g., a cloud provider, to act on its behalf, typically with a subset of its access rights, safely and securely.

This paper deals with delegation across storage clouds. We describe a delegation architecture for on-boarding federation, which allows an enterprise to efficiently migrate its data from one storage cloud provider to another (e.g., for business or legal reasons), while providing continuous access and a unified view over the data during the migration. In our architecture a user delegates a subset of his access rights on the source and destination clouds to an on-boarding federation layer on the destination cloud. This enables on-boarding to occur in a safe and secure way, such that the on-boarding layer has the least privilege required to carry out its work. We evaluate the security implications of delegation that need to be taken into account for on-boarding. We also show how the delegation architecture can be implemented using the Security Assertion Markup Language.

Key words: Storage Cloud, Federation, Delegation, SAML.

1. Introduction. Existing storage clouds do not provide true data mobility as well as adequate mechanisms for allowing efficient migration of their data across providers. This of problem of "data lock-in" is considered to one of the top ten obstacles for growth in Cloud Computing [1]. In a recent paper Vernik et al. [30] present an architecture for *on-boarding federation* to deal with this problem. On-boarding federation allows an enterprise to efficiently migrate its data from one storage cloud provider to another (e.g., for business or legal reasons), while providing continuous access and a unified view over the data over the course of the migration. On-boarding is provided through a federation layer on the new destination cloud by setting up a relationship between its containers and the containers on the old source cloud. Once the relationship is set up, the on-boarding layer is responsible to carry out the migration on behalf of the user, reading objects from the old source cloud and writing objects to the new destination cloud. This layer acts on behalf of the user and requires authorization from the user to act in his/her name with the old and new providers.

A *delegation* mechanism empowers one actor, e.g., an end user, to authorize another actor, e.g., a cloud provider, to act on its behalf, typically with a subset of its access rights, safely and securely. In this paper we show how to employ delegation for on-boarding federation. In particular, when a user sets up an on-boarding relationship between a container in the new and old clouds, the user also delegates a subset of his/her access rights to the federation layer of the new cloud. This subset should include the minimum rights needed for the federation layer to on-board objects of the old container. The delegation mechanism is also secure; it ensures that no other entity except the federation layer can employ the rights delegated to it.

In this paper, we consider two popular standards for delegation, OAuth 2.0 [28] and the Security Assertion Markup Language (SAML) [24]. Motivated by the better security of SAML, we developed an architecture allowing to use it for delegation between the two clouds involved in on-boarding federation. We detail our solution implemented in the context of the VISION Cloud [13], which is an EU-funded project developing advanced features for storage clouds. We provide details of the delegation API and the SAML assertions used to implement it.

The paper is organized as follows. Section 2 overviews the VISION Cloud architecture and Sect. 3 describes the principles of cloud access control and delegation. Section. 4 describes the difference between single sign-on and delegation, and then presents and compares two techniques for web delegation, OAuth 2.0 and SAML. Sect. 5 presents use cases for delegation in VISION Cloud. We present our delegation architecture for on-

*1)Dept. of DICIEMA, University of Messina, Italy 2)IBM Haifa Research Lab, Israel 3)Engineering Ingegneria Informatica SPA, Italy

boarding and its implementation using SAML in Sect. 6. Sect.s 7 and 8 review the related work and conclude. Finally, the appendices detail the delegation API and the SAML assertions implementing it.

2. A brief overview of VISION Cloud. The VISION Cloud architecture is designed to support tens of geographically dispersed data centers (DCs, see Figure 2.1), where each DC may contain tens of clusters each with hundreds of storage-rich compute nodes. An object consists of data and metadata, where the data can be of arbitrary size and type, and the metadata can include arbitrary key value pairs, typically describing the content of the object. Object data cannot be updated, but an entire new version of an object can be created.

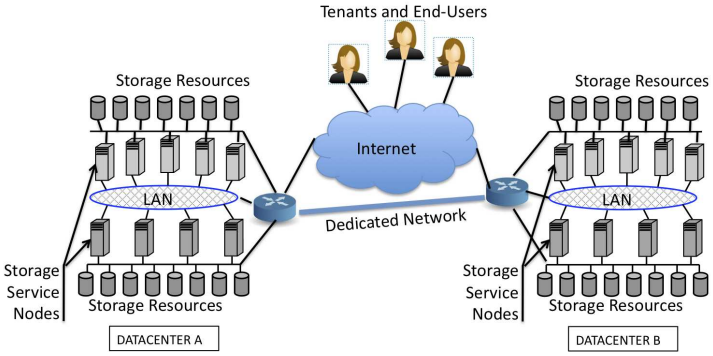

FIG. 2.1. *VISION Cloud Reference Framework: the physical view*

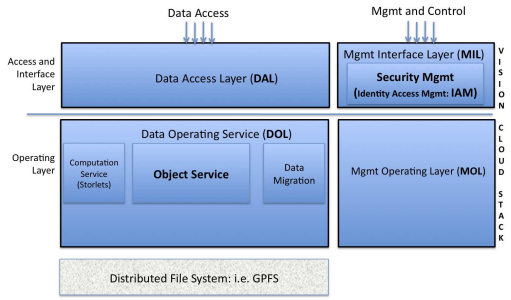

FIG. 2.2. *Simplified Stack of VISION Cloud: the logical view*

In contrast, new metadata can be appended to an object and updated over time. Data objects are grouped in containers, and each data object has a unique name within a container. Containers provide context and are used for purposes of data management, isolation, and placement (i.e., containers are the minimal units of placement and can not be split across clusters). Metadata can also be associated with containers.

The account model includes tenants and users. A tenant subscribes to cloud storage services and usually represents a company or an organization. A tenant may have many users. A tenant administrator creates user accounts and manages them. A user has an identifier and may have credentials allowing him to authenticate himself to the cloud; finally, a user may create containers and data objects in them.

A user is the entity that actually uses storage services, and may refer to a person or to an application. A user belongs to one and only one tenant, although a person might own a user account in more than one tenant.

Figure 2.2 shows a simplified view of the VISION Cloud stack. On the left side, the *Data* layers are depicted wheres on the right side there are represented the *Management* layers. The picture highlights the two main elements (Object Service and Identity Access Manager, see the text of labels in bold) involved in this paper.

3. Identity and access management systems for storage clouds. Most storage clouds provide RESTful interfaces to allow manipulation of resources with standard HTTP methods. Thus, we address access control methodologies in the context of web technologies, considering two stages: authentication and authorization. Authentication is the phase responsible for verifying the identity of a user needing to access web resources. Authorization is the phase responsible for verifying the operation (i.e., read, write, delete, etc.) that users may make on web resources. We use the term **Identity and Access Management (IAM)** to identify the systems able to perform the phases mentioned above. In federated storage cloud scenarios the IAM systems may assume more complex configurations. There are three identity management architectures that are relevant for federation:

- **One Shared IAM:** In this set up the two federated storage clouds use the same internal IAM. This architecture is possible when there are two Clouds deployed on the same public infrastructure. For example, it could correspond to two different OpenStack Swift [27] deployments sharing the same Keystone [10] infrastructure. Any authentication protocol can be used in this set up.
- **One External IAM:** In this case each cloud customer (tenant) has its own IAM, defining an *identity and access domain* that establishes a trust relationship with both federated Clouds. This architecture

allows the user to authenticate with a single set of credentials for applications residing at different cloud and non-cloud systems.

- **Two IAMs:** Each cloud may have its own internal IAM. For example, let's consider two clouds named A and B, in which A needs to access resources hosted in B. In this case the Cloud A should have the credentials on IAM of B to authenticate against the Cloud B.

In the case of *One IAM* (shared or external), the access control is not particularly complex, because the two clouds rely on the same IAM. However, in the case of *Two IAMs* the scenario is much more complex. Every cloud, having an internal IAM, needs to overcome the issues of authentication and authorization of users belonging to different administration domains. Delegation is a common way of overcoming the problems while preserving the proper privileges of users of each cloud. Before discussing our solution to the problem of two IAMs, we describe the concept of delegation (see Sect. 3.1) and compare the existing web delegation technologies (see Sect. 4).

3.1. Introduction to Delegation. Delegation provides the capability for a user (*U1*) to delegate a subset of his access privileges to another user or process (*U2*). *U1* is called *delegator* while *U2* is the *delegate*. Both the users keep their own identities, but *U2* obtains a delegation document signed by *U1* and stating that *U2* is authorized to act on behalf of *U1* for certain operations. Note that *U2* does not obtain the identity of *U1*, i.e., *U2* **does not impersonate** *U1*, rather *U2* is explicitly authorized to perform certain actions on behalf of *U1*. An example taken from everyday life, would be a person going to a public office to get a document on behalf of another person. In this case the human delegate shows his own ID card and a *Power of Attorney*, the delegation document, signed by the delegator, and a copy of the delegator's ID card.

In computer science the concept is identical, in particular:

- The *delegator* provides the *delegate* with an electronic delegation document, digitally signed, containing the details of the delegation (permitted operations, possibility to transfer the delegation, etc.)
- The *delegate* provides his credentials and the delegation document in order to use the delegation and obtain access.

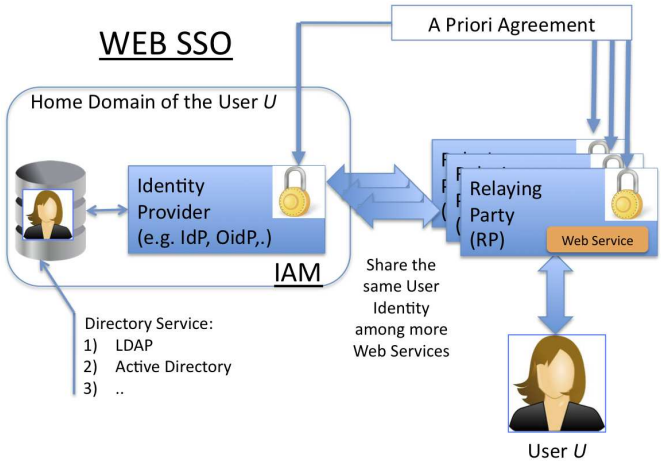

FIG. 4.1. *An example showing Web Single Sign-on*

4. Web Single Sign-on vs. Access Delegation. Single Sign-On (SSO) is an identity federation mechanism removing the need to have multiple accounts on different Relying Parties. Other terms for this party include Service Provider (*SP*). The user can login once to one provider and then access multiple web services. Figure 4.1 shows a typical SSO system, consisting of a *Relying Party* (*RP*), an *IAM* (Identity Provider or OpenID Provider - *IdP*, *OIdP* - *relying on an internal Directory Service*), and the *end-user* (here identified with the User U). User U has an identity registered in the IAM and needs to access a certain web resource on RP:

1. The user is redirected to IAM for the authentication (the authentication **must** be performed on the home domain).
2. If the authentication has succeeded, he/she is redirected to RP for obtaining the requested resource.

Examples of protocols providing Web SSO are SAML [24], OpenID [20], CAS [2]. SAML, OpenID and CAS work at different levels: the first two protocols provide inter-domain Identity Federation (with some difference), while CAS provides only SSO on a single domain.

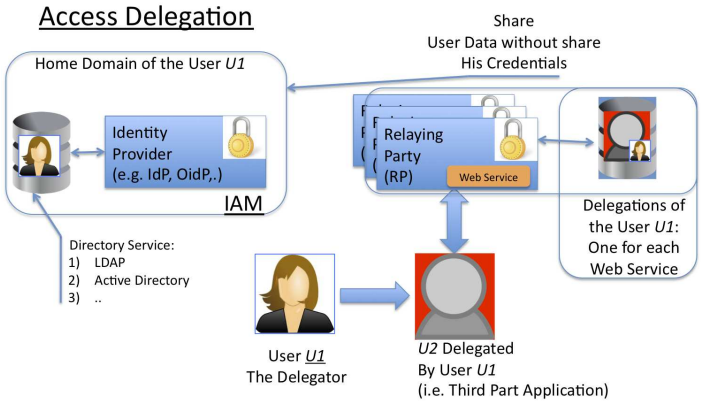

FIG. 4.2. *An example showing the Access Delegation procedure*

The Access Delegation procedure is conceptually different than identity federation in SSO. It is the mechanism employed when a consumer application (e.g., Instagram) wants to use Twitter(Tw) or Facebook (Fb) to post a Tweet or write on a Facebook wall on behalf of an end user. Let's assume an *end-user*, that after downloading a consumer application from mobile market place, decides to exploit the application for interacting with Tw/Fb resources. At the early stage the user gains the access to Tw/Fb through the consumer application, hence the application itself will continue to work without any user supervision. Specifically, the user gains temporary access to Fb/Tw, obtaining *special* tokens necessary for the consumer application. These tokens are uniquely bound and tailored to the application, and they can allow limited operations on behalf of the user. Figure 4.2 depicts all parts composing a typical Access Delegation system. In the figure the Service Provider (SP) can be Fb and/or Tw services, whereas the Third Part Application is equivalent to a consumer application (e.g., Instagram, see the icon avatar in the figure.) For example, using this delegation model the Instagram application is able to post pictures on Facebook on behalf of the end-user. To summarize:

- Web SSO provides a mechanism to *Share the same User Identity* among multiple Web Services. Hence, SSO allows logging in once and then accessing several web services.
- Access Delegation provides a mechanism to *Share User Resources* without sharing credentials.

Access delegation has been widely used in multiple applications implemented with well know protocols like OAuth [28]. However, when considering a setup with delegation across two clouds OAuth protocols have the limitation that they assume the existence of an IAM server shared between the clouds. Another way to setup delegation is using a modified version of the SAML protocol. SAML was originally designed for allowing SSO, but with several new specifications it is now able to provide delegation as well, e.g., through SAML 2.0 Condition to Delegate [22]. The next section introduces OAuth and SAML 2.0 Condition to Delegate solutions, highlighting the advantages and the disadvantages of each.

4.1. OAuth 2.0. The RFC 6749, defined by the IETF, describes the OAuth 2.0 Authorization Framework, and it specializes the existing features of OAuth 1.0 (for further details see the reference [7]). In particular the OAuth 2.0 Authorization Framework allows a third-party application to achieve limited access to an HTTP service. Looking at the OAuth 2.0 Framework it is possible to identify the following actors: *Client*, *Resource Owner*, *Authorization Server*, and *Resource Server*. The **Client** is the application that needs to access protected resources on behalf of the user **Resource Owner** that is the user owning the protected resources that the client needs to access. **Authorization Server** is the server that provides access tokens to the client after the user

has been authenticated and has been granted access to his resources. **Resource Server** is the server managing the user protected resources.

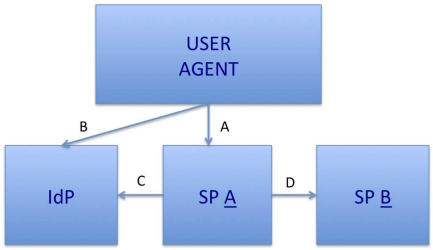

FIG. 4.3. *SAML Condition to Delegate Abstract Protocol Flow*

4.2. SAML 2.0 Condition to Delegate. SAML was introduced by the OASIS consortium following the standardization of XML docs (i.e., SAML [24], XACML [31]). The SAML 2.0 Condition to Delegate (SAML2Del) specification [22] shows how to use the SAML 2.0 protocol for delegation. In particular SAML2Del describes the following delegation-like scenarios:

- **Proxying** where an intermediate identity provider issues an assertion to a relying party on the basis of an assertion issued to it. Proxying is a gateway in which the subject of the assertion is presumed to directly interact with each party.
- **Impersonation** where an entity acting on behalf of an assertion subject is able to obtain and use an assertion indistinguishable from an assertion that can be issued directly to the subject.
- **Forwarding** where an assertion is directly reused by an intermediary to impersonate a subject from whom an assertion was obtained. Forwarding is a form of Impersonation, where the assertion is not modified.
- **Delegation** goes beyond the forwarding scenario by adding information to the assertion that explicitly identifies all the parties through which a transaction flows.

Figure 4.3 depicts how delegation works using SAML. The User Agent (called the Client in the OAuth 2.0 model) gains access to Service Provider A (SP A), using its credentials stored in the Identity Provider (IdP). The SP A also obtains a SAML Assertion (identifying itself to the IdP using its credentials); hence, it can access the resource on SP B. The SAML Assertion of SP A is different respect to the User Agent version; it is modified. In the Figure, all steps characterizing the protocol are shown using the $A - D$ letters.

4.3. Comparison of OAuth 2.0 with SAML 2.0 Condition to Delegate. "OAuth 2.0 Threat Model and Security Considerations" [17] details the main threats and attacks on OAuth 2.0 as well as countermeasures to overcome them. One reason for the weakness of OAuth 2.0 is its evolution from OAuth 1.0 along with the adoption of incremental improvements that have exposed the system to possible flaws, such as:

a) compromising the communication among the parties;
b) obtaining client secrets; and
c) eavesdropping on access tokens.

The OAuth 2.0 protocol provides a greater degree of flexibility with respect to OAuth 1.0, especially in the way it can be applied and the use cases that it addresses, which may come at the price of security [17]. SAML is a much more mature framework conceived for many security purposes, in which the exchange of XML "assertion" guarantees a high degree of security. Especially if we consider the possibility offered by SAML to sign all communications with X509 certificates embedded into XML tags (see the XML Signature and XML Encryption Native Support of SAML 2.0 [23]). For example, XML Signature and XML Encryption help to avoid threats like *a*) and *b*) respectively. Given the greater security of the SAML, we chose it for our work, and in the next sections we describe our adoption of SAML for delegation, presenting detailed examples of SAML Assertions. Note also, that with the goal of improving the current OAuth 2.0 solution and reducing its weaknesses a working group of the IETF has recently released the draft: *SAML 2.0 Profile is used for OAuth 2.0 Client Authentication and Authorization Grants* [25], which describes a hybrid approach adopting SAML to strengthen OAuth 2.0.

5. VISION Cloud delegation scenarios. VISION Cloud addresses various use cases from different industries such as telco, media, healthcare and enterprise applications. Notably, all of them require some sort of delegation. Below, we describe some of the scenarios.

- **Delegation for Federated Resources** VISION Cloud also addresses scenarios where one cloud can *rent* external storage resources from other, foreign, clouds. The cloud needing these resource is responsible for accessing data objects in the new cloud on the behalf of the end user. The Delegation mechanism allows solving the access control problem of this scenario, where, as described above each cloud may have its own IAM.
- **Delegation for Storlets** VISION Cloud uses computational elements (special Agents called Storlets) for performing on-demand computations close to the data physical location. Access delegation is required to properly control a storlet's access to a user's resources on his behalf (e.g., storlets that perform data analytics or statistics should have the proper permissions from the resource owners).
- **Delegation for On-Boarding** On-boarding is a process in which a cloud customer migrates his data from one storage cloud (a New Cloud, i.e., Cloud A) to another cloud (an Old Cloud, i.e., Cloud B). Delegation mechanisms allow to grant the on-boarding component (termed the *Federator*) sufficient, but limited access rights to the customer resources.

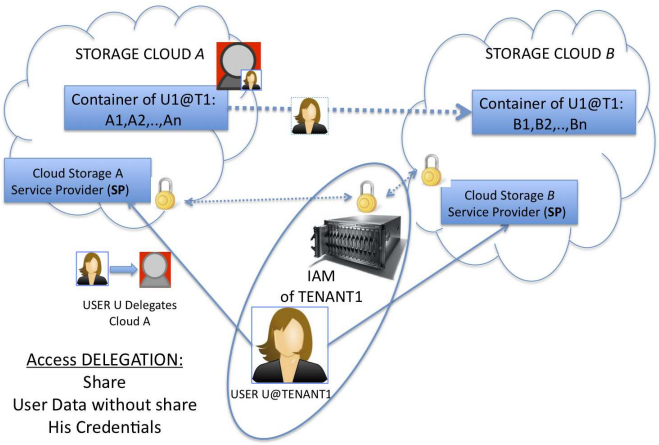

FIG. 5.1. *Delegation for VISION Cloud in On-Boarding Scenario*

In the remainder of the paper we use this last use case to illustrate delegation for on-boarding federation. Below we present the on-boarding federation system (see Figure 5.1). It was implemented as part of the VISION Cloud, but can also be easily added to other cloud systems wishing to provide a service for migration (on-boarding) from other providers. The new provider (Cloud A) is responsible for moving the customer data to itself from the old cloud provider (Cloud B). Furthermore, following the federation set up, applications and users begin accessing their data through the new cloud, which provides instant access to customer data remaining in the old cloud. Thus, the applications that access the migrated data are not influenced by on-boarding and can work transparently. The old cloud is assumed to be unaware of the on-boarding and is not required to introduce any modification. *Migrating data via on-boarding federation directly between the clouds leads to a significant savings in time and cost [30].* From the technical standpoint, the on-boarding architecture specifies the following three primary flows:

1. *On-boarding set-up:* An on-boarding relationship between a container in the old cloud and a container in the new cloud is set up and persisted through the protected metadata of the container on the new cloud.
2. *Direct access:* Once the relationship is set-up, all client's applications may start to immediately access the objects of the old container through the new cloud. When a client accesses an object on the new cloud that has not yet been on-boarded, the Federator module gets the object from the old cloud and puts it in the new cloud on the behalf of the end user.

3. *Background on-boarding:* The Federator component creates background jobs on the the cloud that fetch objects from the container in the old cloud and copy them to the container in the new cloud. These jobs run when the resource utilization (e.g., CPU and network) in the new cloud is low so that they do not interfere with the normal operation of the cloud. These jobs also need authorization from the user to access the old cloud and depending on the architecture may also need authorization the write the objects on the new cloud.

The delegation architecture to provide the authorization required for the direct and background on-boarding is described in the next section.

6. VISION Cloud on-boarding delegation architecture. We first describe the flow for the SAML-based delegation mechanism that we have chosen, and then show how we apply it for on-boarding.

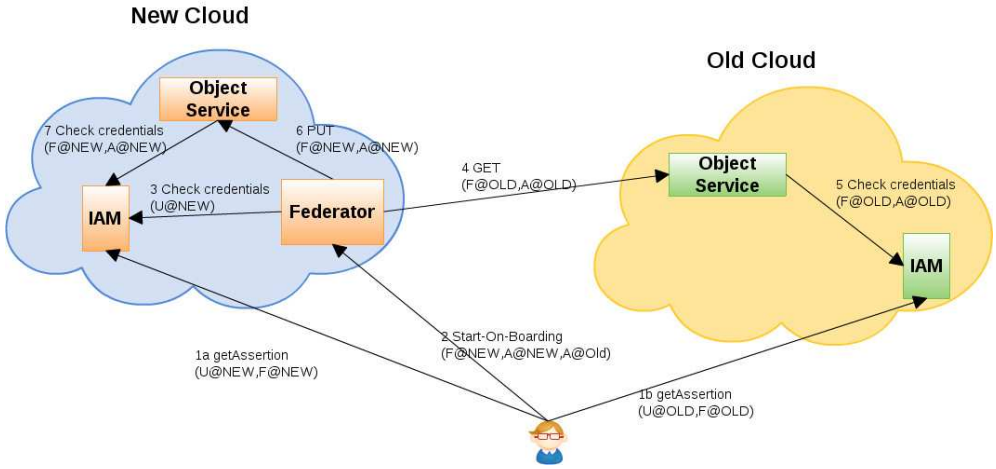

FIG. 6.1. *Delegation for on-boarding in VISION Cloud*

Based on the comparison presented in Sect. 4.1 we chose to implement the delegation using SAML2Del. In this protocol, the delegation document is a signed SAML assertion containing the details of the delegation, held in a specific field called *Condition*. To describe the delegation flow, we consider a simple setup with single IAM and two accounts, U1 and U2, which correspond to the user in the old and new clouds respectively, where a user can also be a process or an infrastructure.

The access flow is as follows:

1. U1 logs into the IAM using her credentials and asks to generate a signed delegation assertion stating that U2 is authorized to perform, for example, GET operations on behalf of U1 for one day.
2. U1 provides the assertion to U2.
3. U2 logs into the IAM using her credentials, provides the assertion and is authorized to perform GET operations for the entire day.

In the on-boarding federation scenario, the Federator reads the objects on behalf of the user U1@Old in the Old Cloud, and writes them on the behalf of the user U1@New in the New Cloud, where @Old and @New correspond to the different identity management servers of the two clouds.

6.1. On-boarding Delegation Flow. Figure 6.1 shows the on-boarding delegation flow between two VISION Clouds: the New and the Old, where each cloud has its own IAM. Each IAM incorporates an Identity Provider (IdP) and a Service Provider (SP) in order to carry out the SAML protocol. The figure shows a Federator component which can be introduced either independently or as an added-value feature of the New Cloud. Each cloud also has an Object Service, which carries out the basic data access operations such as GET/PUT of objects and containers.

The Federator has an identity in both clouds. In particular, let $F@OLD$, denote the federator identity (F) in the IAM of the Old Cloud and let $F@NEW$ denote its identity (F) in the IAM of the New Cloud. The user (U) that requests the on-boarding also has an identity on each cloud, in particular, $U@OLD$ (in the Old

Cloud) and $U@NEW$ (on the New Cloud). All the identities must be registered with suitable credentials (e.g. username/password) in their corresponding IAMs.

The delegation flow is modeled using two delegations, in particular: (1) $U@OLD$ delegates $F@OLD$ to GET her objects from her container on Old Cloud; and (2) $U@NEW$ delegates $F@NEW$ to PUT objects on her container in New Cloud.

The following operations are performed during the delegation flow for on-boarding.

1. The user logs in and obtains delegation assertions for both clouds:
   (a) Delegation assertion ($A@NEW$), based on the credentials of $U@NEW$ and the userid of $F@NEW$.
   (b) Delegation assertion ($A@OLD$), based on the credentials of $U@OLD$ and the userid of $F@OLD$.
2. The user starts the federation process, providing the following parameters:
   (a) $U@NEW$ - credentials to activate the Federator.
   (b) $A@NEW$ - the assertion ID delegating $F@NEW$ to ask for PUT operations on behalf of $U@NEW$.
   (c) $A@OLD$ - the assertion ID delegating $F@OLD$ to ask for GET operations on behalf of $U@OLD$.
3. The Federator authenticates $U@NEW$ through the IAM of the New Cloud.
4. The Federator performs a GET operation to the Old Cloud using $F@OLD$ credentials as delegate of $U@OLD$.
5. The Object Service of the Old Cloud checks the credentials and the delegation with the IAM of the Old Cloud.
6. The Federator PUTs the retrieved object to the New Cloud using $F@NEW$ credentials as Delegate of $U@NEW$.
7. The Object Service of the New Cloud checks the credentials and the delegation with the IAM of the New Cloud.

6.2. Delegation Advantages. The Federator acts as a canonical user, authenticating itself for each REST request (GET from Old Cloud and PUT to New Cloud). The delegation protocol allows it to access the user resources without possessing the user credentials or requiring an explicit trust relationship with the Old Cloud. The assertion IDs are inserted into the REST messages and they are cached by the IAM for increasing the performances (as below). In addition to preserving the security, this scheme has the advantage of an easy combination with other authorization methods used in the underlying object store system. For example, VISION Cloud uses Access Control Lists (ACLs), which are associated with every resource, such as object or container. ACLs are configured with a RESTful API as specified by CDMI, where the syntax is NFSv4 compatible allowing compatibility with traditional file systems. Since ACLs are distributed with the resource, they provide an efficient way of setting fine-grained access control with per object/container permissions. Notably, by granting the Federator specific but limited permissions to act on the behalf of the end user, the described delegation allows to properly preserve all of the permissions set by ACLs. Specifically, after the Object Service in Figure 6.1 checks the Federator credentials and assertion (Step 7), the system drops the privileges to those defined in the assertion (the identity and the delegated role granted to the Federator by the end user). Due to the mechanism of role delegation the Federator can securely access the resources without the need to specifically add it to the ACLs.

7. Related Work. Below, we review the models for federation between cloud operators and then focus on works for federation of access control for identity federation and delegation.

Usually federation consists of the establishment of a trust context between parties with the purpose of benefiting of business advantages. According to the view of the VISION Cloud project, Cloud federation may represent a compelling business model for SMEs, where many stakeholders (i.e., Cloud providers, tenants and customers) interact with each other for creating new opportunities and satisfying even more needs [12]. There are four basic capabilities that characterize each entity of the federation, which "keeps authority about the information passed to the other entities of the federation" and "has authority to create a global view of the data that is available among all the entities of the federation". An entity of the federation "is not forced to perform tasks of another entity of the federation" and "can autonomously decide to enter or leave the federation". Examples of such systems were presented by Tordsson et al. [29], Kurze et al. [14] as well as the recent Colony system, that federated several Openstack Swift deployments in a Colony by introducing a Swift Dispatcher component, prefixing the container names. Unlike this work, we do not assume any naming conventions or

architectural components common to the two clouds. The federation described in [6] is aimed at computation management (consolidation of VMs among Clouds). It shows similarities to the work done by Celesti et al. [3] and Rochwerger et al. [21], where the federation problems for data management between several IaaSs are addressed. In the latter case the authors described how to elastically enlarge in a transparent way, the physical resource of cooperating IaaSs.

From the security and privacy standpoint, the cloud has not kept pace with the enormous volumes of user identities that network administrators must manage and secure. An identity fabric that links multiple applications to a single identity would address this problem, enabling full-scale cloud adoption as it is highlighted in the *Architecting a Cloud-Scale Identity Fabric* (see [18]). The authors in [4] identified the services and APIs are necessary to be realized for accessing cloud resources in a useful and focused way. They range from: *Federated Identity*, to *Delegation Of Authority*, and *Levels Of Assurance*, *Attributes*, *Access Rights*, till *Authorization*. In this work, the authors described all APIs they introduced, and applied them to a real cloud middleware, the Eucalyptus S3 Service. This work explains how the researchers have used these proof-of-concept APIs and how to exploit and match them with the existing services of Kent University.

The Security Assertion Markup Language (SAML) became a popular technology for solving issues related to the federation needs. Indeed many existing works provide solutions for a variety of scenarios that are based on the SAML mechanism and its capabilities. These solutions do not solve all the cloud federation problems. The federation problem in cloud computing is greater than the one in traditional systems.

The first practice of cooperation between providers was about sharing the network traffic. A few years later service federation over the Internet has become a well established approach: it had to be supported by a mechanism for trusting identities across different domains, which is identity federation. The latest trend to federate identities over the Internet is represented by the Identity Provider/Service Provider (IdP/SP) model [16], supported by digital certificates.

An important example of a popular implementation of SAML is Shibboleth [26]. Unfortunately, most systems still lack important capabilities required for the federation among different cloud providers. The main constraint of the existing federation solutions is that they are designed for static environments requiring a priori policy agreements, whereas clouds are dynamic and heterogeneous environments which require particular security and policy arrangements.

Interoperability in federated heterogeneous cloud environments is addressed in [15], in which the authors propose a trust model where the trust is delegated between trustworthy parties satisfying certain constrains. The approach in VISION Cloud considers also the issue with the data location and protection with fine-grain ACLs. Pearson at al. [19] also introduce a privacy manager, taking care of data compliance according to the laws of different jurisdictions.

Huang et al. present [9] an Identity Federation Broker for service clouds. They address federation in the context of SaaS services in which SAML represents the basic protocol. They make use of a trusted third party as a trust broker to simplify the management of identity federation in a user centric way.

A recent work called *FACIUS* describes the use of SAML for Non Web-Based Services [11]. The authors make an assessment of a system accomplishing a real implementation of SAML with SSH. In particular they have reported evaluations with respect to requirements, performance, security, and legal aspects.

The works presented above show various SAML applications and justify its adoption in our solution. However, the they consider it in a different aspect which is not suitable for our aims. Here, in addition to presenting standard SAML flows, we discuss a slightly modified version allowing to simplify the on-boarding federation.

When considering the delegation technologies, protocols leveraging the Public-Key Infrastructures (PKI) was widely adopted grid systems. For example, grids make use of signed certificates for users' identification and jobs submission. In grid terminology, Virtual Organizations (VOs) represent entities that are able to manage certificates, distributing and verifying them, accomplishing a full PKI. Right now, a recent trend for managing credential and delegations in federated cloud scenarios appears to make use of X.509 certificates.

In the direction of managing identity and authorization for Community Clouds using PKI is discussed in [5]. The authors introduce an *identity broker* to bind the Web Single Sign-on to a key-based system. In particular they implemented a solution (*libabac* package) using the Attribute-based Access Control (ABAC) and the role-based trust management (RT): RT/ABAC. The *libabac* uses X.509 as a transport. RT/ABAC credentials are

X.509 attribute certificates. The RT/ABAC may handle the processing of delegation chains.

A recent and detailed work on delegation is described in the paper entitled: *OAuth and ABE based Authorization in Semi-Trusted Cloud computing, [28].* Here, the authors enhanced the OAuth capabilities using the encryption in attribute-based access control system exploiting metadata. They introduced a complex model in which authentication, authorization, delegation, access trees, new tokens, time slots, user certificates are investigated. Their objective is to provide a complete solution with a high level of flexibility useful in many cloud scenarios.

In spite this progress early storage cloud solutions focused on scalability and had difficulties in achieving the delegation requirements [8]. However, in recent years they are slowly adopting more advanced access control technologies. For example, the OpenStack security and access control component Keystone has recently adopted PKI (see [10]). Keystone uses tokens, which are json documents that contain the required access control information about users, projects, roles and domains. Starting from the Grizzly release of Openstack, these tokens are cryptographically signed based on the X509 standard. Keystone serves as a Certificate Authority (CA) and uses its signing key and certificate to sign tokens. Thanks to PKI, other OpenStack entities can then locally verify these tokens with the public key without the need to contact Keystone.

8. Conclusions and Future Work. In this paper we presented the concept of delegation for on-boarding federation between storage clouds. Our solution is based on the SAML 2.0 Condition to Delegate extension. The added value of this work is in considering all security implications in using the delegation technique for the on-boarding procedure. We also show how this delegation architecture can be implemented using SAML solution. We implemented the solution as part of our work on VISION Cloud, accomplishing RESTful APIs and creating new SAML delegation Assertions. In the future we will evaluate the impact of it on VISION Cloud architecture, analyzing its complexity and performance.

Acknowledgments. The research leading to the results presented in this paper has received funding from the European Union's Seventh Framework Programme (FP7 2007-2013) Project VISION-Cloud under grant agreement number 217019.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia, *Above the Clouds: A Berkeley View of Cloud Computing*, Tech. Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[2] CAS, *Central authentication service,*. http://www.jasig.org/cas, June 2013.

[3] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, *Integration of clever clouds with third party software systems through a rest web service interface*, in Proceedings - IEEE Symposium on Computers and Communications, 2012, pp. 827–832.

[4] D. W. Chadwick and M. Casenove, *Security apis for my private cloud - granting access to anyone, from anywhere at any time*, in Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, CLOUDCOM '11, Washington, DC, USA, 2011, IEEE Computer Society, pp. 792–798.

[5] J. Chase and P. Jaipuria, *Managing identity and authorization for community clouds*, tech. report, Department of Computer Science, Duke University, 2012. Technical Report CS-2012-08.

[6] I. Goiri, J. Guitart, and J. Torres, *Characterizing cloud federation for enhancing providers' profit*, in Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, Jul 2010, pp. 123–130.

[7] D. Hardt, *The OAuth 2.0 Authorization Framework*. RFC 6749 (Proposed Standard), Oct. 2012.

[8] D. Harnik, E. K. Kolodner, S. Ronen, J. Satran, A. Shulman-Peleg, and S. Tal, *Secure access mechanism for cloud storage*, Scalable Computing: Practice and Experience, 12 (2011).

[9] H. Y. Huang, B. Wang, X. X. Liu, and J. M. Xu, *Identity federation broker for service cloud*, in Service Sciences (ICSS), 2010 International Conference on, May 2010, pp. 115–120.

[10] KEYSTONE, *Welcome to keystone, the openstack identity service*. http://docs.openstack.org/developer/keystone, 2013.

[11] J. Kohler, S. Labitzke, M. Simon, M. Nussbaumer, and H. Hartenstein, *Facius: An easy-to-deploy saml-based approach to federate non web-based services*, in Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on, Jun 2012, pp. 557–564.

[12] E. K. Kolodner, A. Shulman-Peleg, D. Naor, P. Brand, M. Dao, A. Eckert, S. Gogouvitis, D. Harnik, M. Jaeger, D. Kyriazis, et al., *Data intensive storage services on clouds: Limitations, challenges and enablers*, European Research Activities in Cloud Computing, D. Petcu and JL Vazquez-Poletti, Eds. Cambridge Scholars Publishing, (2012), pp. 68–96.

[13] E. K. Kolodner, S. Tal, D. Kyriazis, D. Naor, M. Allalouf, L. Bonelli, P. Brand, A. Eckert, E. Elmroth, S. V. Gogouvitis, D. Harnik, F. Hernández, M. C. Jaeger, E. B. Lakew, J. M. Lopez, M. Lorenz, A. Messina, A. Shulman-Peleg, R. Talyansky, A. Voulodimos, and Y. Wolfsthal, *A cloud environment for data-intensive storage services*, in CloudCom, 2011, pp. 357–366.

[14] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze, *Cloud federation*, in Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011), IARIA, Sep 2011.

[15] W. Li and L. Ping, *Trust model to enhance security and interoperability of cloud environment*, in Cloud Computing, Nov 2009, pp. 69–79.

[16] Liberty, *An alliance project*. http://projectliberty.org, 2013.

[17] T. Lodderstedt, M. McGloin, and P. Hunt, *OAuth 2.0 Threat Model and Security Considerations*. RFC 6819 (Informational), Jan. 2013.

[18] E. Olden, *Architecting a cloud-scale identity fabric*, Computer, 44 (2011), pp. 52–59.

[19] S. Pearson, Y. Shen, and M. Mowbray, *A privacy manager for cloud computing*, in Cloud Computing, Nov 2009, pp. 90–106.

[20] D. Recordon and D. Reed, *Openid 2.0: a platform for user-centric identity management*, in Proceedings of the second ACM workshop on Digital identity management, DIM '06, New York, NY, USA, 2006, ACM, pp. 11–16.

[21] B. Rochwerger, S. Naqvi, C. Ponsard, J. Latanicki, P. Massonet, and M. Villari, *A monitoring and audit logging architecture for data location compliance in federated cloud infrastructures*, in IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, 2011, pp. 1510–1517.

[22] SAML-DEL, *V2.0 condition for delegation*. http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-delegation-cs-01.pdf, 2013.

[23] SAML-ENHANC, *2.0 enhancements*. http://saml.xml.org/saml-2-0-enhancements, 2007.

[24] SAML-OASIS, *V2.0 technical*. http://www.oasis-open.org/specs/index.php, Jan 2013.

[25] SAML-OAUTH, *Saml 2.0 profile for oauth 2.0 client authentication and authorization grants", note="http://datatracker.ietf.org/doc/draft-ietf-oauth-saml2-bearer, 2013.*

[26] Shibboleth, *System standards*. http://shibboleth.internet2.edu/, Jan 2012.

[27] SWIFT, *Welcome to Swift's documentation*, June, 2013. http://http://docs.openstack.org/developer/swift/.

[28] A. Tassanaviboon and G. Gong, *Oauth and abe based authorization in semi-trusted cloud computing: aauth*, in Proceedings of the second international workshop on Data intensive computing in the clouds, DataCloud-SC '11, New York, NY, USA, 2011, ACM, pp. 41–50.

[29] J. Tordssona, R. S. Monterob, R. Moreno-Vozmedianob, and I. M. Llorente, *Optimized placement of a computational cluster across multiple clouds*.

[30] G. Vernik, A. Shulman-Peleg, S. Dippl, C. Formisano, M. Jaeger, E. Kolodner, and M. Villari, *Data on-boarding in federated storage clouds*, in IEEE CLOUD 2013 IEEE 6th International Conference on Cloud Computing June 27-July 2, 2013, Santa Clara Marriott, CA, USA (Center of Silicon Valley), 2013.

[31] XACML, *Cross-enterprise security and privacy authorization (xspa) profile of xacmlv2.0 for healthcare version 1.0*. http://www.oasis-open.org/committees/document.php?document_id=34164&wg_abbrev=xacml.

Appendix A. On-Boarding Delegation details: REST API.

In this section we present the API of VISION Cloud for delegation, first a description of the parameters and then some details of the RESTful API for creating and using a delegation.

A.1. API parameters. Table A.1 shows the parameters to be provided by an end-user (the delegator) during the interaction with his IAM to create a delegation document and the delegation token representing it to be granted to the delegate. The delegator provides his credentials, and describes the capabilities and the actions he wishes to delegate, including a limitation of the delegation to certain containers, and identifies the delegate.

Name	Type	Mandatory/Optional	Description
delegatorUsername	JSON String	mandatory	Name of the delegator user.
delegatorPassword	JSON String	mandatory	Password of the user.
delegatorTenant	JSON String	mandatory	Tenant of the user.
delegatedId	JSON String	mandatory	Identifier of the delegated user.
delegatedTenant	JSON String	mandatory	Name of the delegated tenant.
delegatedRoles	JSON Array	mandatory	Roles of the delegator (to be delegated).
delegatedActions	JSON Array	mandatory	Actions of the delegator (to be delegated).
delegatedContainer	JSON String	mandatory	Name of the container (to be delegated)

TABLE A.1
Delegation capabilities provided to a delegated user and/or software component *(e.g., the Federator).*

A.2. RESTful API for on-boarding delegation. In VISION Cloud storage related operations (create/read/update/delete) are invoked through a RESTful API over standard HTTP. A token for a delegation assertion is also obtained through a RESTful request. In particular, Listing 1 shows a request to create a delegation assertion and return a token for it, e.g., as required in steps 1a and 1b of Figure 6.1.

LISTING 1
HTTP - User Delegation Request

```
Request:
POST <root-uri>/visionIAM/requestDelegation
       Accept: application/json
       Content-Type: application/json
       {
              "delegatorUsername":"delegator_username",
              "delegatorPassword":"delegator_password",
              "delegatorTenant":"delegator_tenant",
              "delegatedId":"delegated_identifier",
              "delegatedTenant":"delegated_tenant",
              "delegatedRoles":["role1","role2",...,"roleN"],
              "delegatedActions":["action1","action2",...,"actionN"],
              "delegatedContainer":"delegated_container"
       }
```

If the delegator is authorized to perform a delegation for the required roles and operations, the response is a 200 OK and contains the the delegation token. Such a response is shown in Listing 2. If the delegator is not authorized then the response is a 400.

LISTING 2
HTTP - IAM Delegation Token Released

```
Response:
HTTP Status
```

```
HTTP/1.1 200OK
Content-Type: application/json
{
"delegationToken":"NGFiYWVmOTctMmY0MS00MTgyLWFkYTItODc0M2EyMDA1MDZi"
}
```

A RESTful request on VISION Cloud, e.g., a GET request to read an object, contains an authorization header holding the credentials (username/password) for the user making the request. The VISION Cloud Identity Management System uses these credentials to authenticate the user. The same header is used for a request on VISION Cloud from a delegate; in this case the header contains the credentials of the delegate and the delegation token. Listing 3 shows an authorization header containing a delegation token.

LISTING 3
HTTP - Delegated Access

```
Request:
GET <root-uri>/visionIAM/authenticate HTTP/1.1
Authorization: DEL base64Encoded{username@tenantName:password:delegationToken}
```

In SAML all messages are signed with X509 certificates, to avoid any kind of attack when managing identities, roles and actions (Authentication and Authorization phases). The signature is done with valid (recognized) certificates and thus is not exploitable for an attack. Section. B briefly describes these SAML security capabilities.

Appendix B. SAML Assertion for On-Boarding Delegation.

FIG. B.1. *SAML Assertion used for the VISION Cloud On-Boarding Delegation scenario*

This section highlights the various parts comprising a SAML Assertion used for the on-boarding delegation scenario. As can be seen, the assertion is an XML-based container of information. In general all SAML assertions are associated with a subject (< *Subject* > element, see Listing 6. The standard defines three statements: *Authentication Assertion*, *Attribute Assertion* (see Listing 7), and an *Authorization Decision Assertion*. In SAML there are also *Signatures*, *Conditions* (see Listing 6) and *Issuers*. The prefix "saml" represents the basic namespace for a SAML V2.0 assertion (see Listing 4). In our case the SAML Assertion has 6 parts as depicted in Figure B.1. In all listings reported below we highlight the flexibility of the SAML standard as well as its strong security.

Listing 4
SAML - Root container and Issuer

```
<?xml version="1.0" encoding="UTF-8"?>
<saml2:Assertion
    xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
    ID="_9a5a551aeb58718ed0076e31d1cf510b" IssueInstant="2013-10-29T10:42:25.908Z"
    Version="2.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
......
</saml2:Assertion>
------
  <saml2:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
  https://identityProvider.hostname/idp
  </saml2:Issuer>
------
```

Listing 5
SAML - Signature parts

```
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
           <ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
             <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"
                  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"/>
             <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"
                  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"/>
             <ds:Reference URI="#_8f1c03bb-bcb1-48eb-8a52-33c19e8a5d66"
             xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
               <ds:Transforms xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                  <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature
                       " xmlns:ds="http://www.w3.org/2000/09/xmldsig#"/>
                  <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" xmlns:ds="
                       http://www.w3.org/2000/09/xmldsig#">
                    <ec:InclusiveNamespaces PrefixList="del xs" xmlns:ec="http://www.w3.org
                         /2001/10/xml-exc-c14n#"/>
                  </ds:Transform>
               </ds:Transforms>
               <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" xmlns:ds="
                    http://www.w3.org/2000/09/xmldsig#"/>
               <ds:DigestValue xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...
               </ds:DigestValue>
             </ds:Reference>
           </ds:SignedInfo>
           <ds:SignatureValue xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...
           </ds:SignatureValue>
           <ds:KeyInfo>
             <ds:X509Data>
               <ds:X509Certificate>...</ds:X509Certificate>
             </ds:X509Data>
           </ds:KeyInfo>
      </ds:Signature>
```

Listing 6
SAML - Subject and Conditions

```
1
2  <saml2:Subject>
3        <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"
4        NameQualifier="https://identityProvider.hostname/idp"
5        SPNameQualifier="https://serviceProvider.hostname/sp">
6        _a36102490a9b9c196255ff86ee1a052f</saml2:NameID>
7            <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
8               <saml2:SubjectConfirmationData>
9                 <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
10                  <ds:X509Data>
11                    <ds:X509Certificate>...</ds:X509Certificate>
12                    <ds:X509SubjectName>...</ds:X509SubjectName>
13                    <ds:X509IssuerSerial>
14                      <ds:X509IssuerName>...</ds:X509IssuerName>
15                      <ds:X509SerialNumber>...</ds:X509SerialNumber>
16                    </ds:X509IssuerSerial>
17                  </ds:X509Data>
18                </ds:KeyInfo>
19              </saml2:SubjectConfirmationData>
20           </saml2:SubjectConfirmation>
21    </saml2:Subject>
22 ---------
23    <saml2:Conditions>
24        <saml2:Condition
25                        NotBefore="2013-10-31T09:40:00.440Z"
26                NotOnOrAfter="2013-10-31T18:40:00.440Z"
27                xmlns:del="urn:oasis:names:tc:SAML:2.0:conditions:delegation"
28                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
29                xsi:type="del:DelegationRestrictionType">
30                <del:Delegate ConfirmationMethod="#sender-vouches"
31                DelegationInstant="2013-10-29T10:42:25.900Z">
32                <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
33                delegated</saml2:NameID>
34            </del:Delegate>
35        </saml2:Condition>
36    </saml2:Conditions>
```

Listing 6 shows the flexibility of SAML in defining *Conditions*. In this case the Assertion is valid for a specific time interval as highlighted in NotBefore and NotOnOrAfter Condition Attributes (see lines 25 and 26). A timestamp allows tracking the creation time of a delegation (see line 31). The *Subject* tag describes all the fields of a X509 user/host certificate necessary for ensuring security (see from line 9 to 18).

Listing 7
SAML - Authentication and Attribute Statements

```
1    <saml2:AuthnStatement AuthnInstant="2011-03-29T16:35:12.440Z" SessionIndex="123456">
2        <saml2:AuthnContext>
3                <saml2:AuthnContextClassRef>
4                              urn:oasis:names:tc:SAML:2.0:ac:classes:PreviousSession
5                </saml2:AuthnContextClassRef>
6        </saml2:AuthnContext>
7    </saml2:AuthnStatement>
8   -------
9    <saml2:AttributeStatement>
10       <saml2:Attribute FriendlyName="delegated_roles" Name="delegated_roles">
11           <saml2:AttributeValue
12               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
13               xsi:type="xs:string">REMOTE_ADVISOR
14           </saml2:AttributeValue>
15       </saml2:Attribute>
16       <saml2:Attribute FriendlyName="delegated_actions" Name="delegated_actions">
17           <saml2:AttributeValue
18               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
19              xsi:type="xs:string">CREATE_OBJECT
20          </saml2:AttributeValue>
21          <saml2:AttributeValue
22              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
23              xsi:type="xs:string">DELETE_OBJECT
24          </saml2:AttributeValue>
25      </saml2:Attribute>
26      <saml2:Attribute FriendlyName="delegated_container"
27      Name="delegated_container">
28          <saml2:AttributeValue
29              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
30              xsi:type="xs:string">7D1FG1
31          </saml2:AttributeValue>
32      </saml2:Attribute>
33      <saml2:Attribute FriendlyName="delegator_username"
34      Name="delegator_username">
35          <saml2:AttributeValue
36              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
37              xsi:type="xs:string">delegator
38          </saml2:AttributeValue>
39      </saml2:Attribute>
40      <saml2:Attribute FriendlyName="delegator_username"
41      Name="delegator_username">
42          <saml2:AttributeValue
43              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
44              xsi:type="xs:string">delegationTenant
45          </saml2:AttributeValue>
46      </saml2:Attribute>
47      <saml2:Attribute FriendlyName="delegated_username"
48      Name="delegated_username">
49          <saml2:AttributeValue
50              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
51              xsi:type="xs:string">delegated
52          </saml2:AttributeValue>
53      </saml2:Attribute>
54      <saml2:Attribute FriendlyName="delegated_tenant"
55      Name="delegated_tenant">
56          <saml2:AttributeValue
57              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
58              xsi:type="xs:string">delegationTenant
59          </saml2:AttributeValue>
60      </saml2:Attribute>
61  </saml2:AttributeStatement>
```

As can seen in Listings 5 and 6, 50% of the SAML structure is devoted to the security definitions (certificates X509, keys, sec algorithm, digests, etc.). These parts are necessary for securing any complex distributed scenario where many actors are involved. The complexity increases if we consider the case of federation among clouds. We remark that other solutions for delegation do not have the same level of accuracy and security in treating and exchanging sensitive data as does SAML. All Actions, Roles and Resources introduced in our implementation are described in the $< saml2 : AttributeStatement >$ statement (see Listing 7). In particular, all fields of table A.1 are reported in the listing from the line 9 to 61.

Edited by: Maria Fazio and Nik Bessis
Received: Nov 2, 2013
Accepted: Jan 10, 2014