Federation Between CLEVER Clouds Through SASL/Shibboleth Authentication

Francesco Tusa, Antonio Celesti, Massimo Villari and Antonio Puliafito
Dept. of Mathematics, Faculty of Engineering, University of Messina
Contrada di Dio, S. Agata, 98166 Messina, Italy.
e-mail: {ftusa, acelesti, mvillari, apuliafito}@unime.it

***Abstract*—Several ICT operators are realizing the advantages of federating cloud providers in order to carry out new business benefits, hence increasing their revenues. Nevertheless, how to achieve a cloud architecture able to perform authentication with other installations is not fully clear. CLEVER is a cloud IaaS middleware conceived with federation in mind. In this paper, we discuss an approach to perform SSO authentication based on an integration between SASL and SAML in a CLEVER environment. More specifically, we describe how to federated the Ejabberd servers, on which the communication of each CLEVER cloud is based, through a Shibboleth authentication.**

***Keywords*-Cloud Computing; CLEVER; Federation; Security; SASL; SSO Authentication; SAML; Shibboleth.**

I. Introduction

Nowadays, most of Cloud providers can be considered as "islands in the ocean of the Cloud computing" and do not present any form of federation. At the same time, a few Clouds are beginning to use the Cloud-based services of other Clouds, but there is still a long way to go toward the establishment of a worldwide Cloud ecosystem including thousands of cooperating Clouds. In such a perspective, the latest trend toward Cloud computing is dominated by the idea to federate heterogeneous Clouds, as it is highlighted in [1]. This means not to think about independent private Clouds anymore, but to consider a new Cloud federation scenario where different Clouds, belonging to different administrative domains, interact each other, sharing and gaining access to physical resources. Cloud Federation is a concept that goes beyond the simple achievements falling into Hybrid Clouds (Private + Public, see [2]).

The US Department of NIST, after the well-known definition of *SaaS, PaaS and IaaS*, is actively working for accelerating Standards to foster the Adoption of Cloud Computing [3]. *Interoperability, Portability and Security* are the main aims of their enforcement. Our work tries to find a solution for the first and third of these aspects seen for federated Cloud scenarios, exploiting CLEVER (see [4]). It is an IaaS Cloud middleware, conceived having in mind federation. The innovation of CLEVER is that its communication system has been designed with the public-subscribe philosophy using the Extensible Messaging and Presence Protocol (XMPP) [5] (see also RFC 6120 [6]). XMPP is an open-standard communications protocol for message-oriented middleware based on XML (Extensible Markup Language). Thus, in CLEVER, each Cloud belongs to a domain managed by an XMPP server. In CLEVER, the way to federate two Clouds is to establish a server-to-server inter-domain communication between the XMPP servers of the involved Clouds.

Cloud federation raises many issues especially in the field of security and privacy. Single Sign On (SSO) authentication is fundamental for achieving security in a scalable scenario such as Cloud federation. However, the Simple Authentication and Security Layer (SASL) [7], i.e., a framework for authentication and data security in Internet protocols, supported by XMPP does not support any SSO authentication mechanism.

In this paper, integrating the SASL with the Security Assertion Markup Language (SAML) protocol [8], we describe an approach to authenticate two or more CLEVER Clouds, discussing an implementation based on Ejabberd [9] and Shibboleth [10]. The paper is organized as follows. Section II describes the state of the art of Cloud middlewares dealing with federation. Section III introduces the CLEVER Cloud middleware, discussing how it supports federation. Section IV describes the technological issues for the SSO authentication achievement during the process of federation establishment. More specifically, a solution based on SASL and SAML is discussed. Section V describes an implementation practice of SSO authentication between CLEVER Clouds using Ejabberd servers and Shibboleth. Section VI concludes the paper.

II. Related Works

Hereby, we describe the current state-of-the-art in Cloud computing analyzing the main existing middleware implementations, emphasizing the federation aspects they attempt to address. Before such a description, it is interesting to point out the results of Sempolinski and Thain work published in 2010 [11]. They provided a comparison among three widely used architectures: Nimbus, Eucalyptus and OpenNebula. They remarked how the projects are aimed at different goals, but a clear convergence is recognizable. The authors posed three main questions, one about who has a complete Cloud computing software stack. It is common in the three architectures that the actual Cloud controller is only a small part of the overall system. The second one is who is really customizable. These are open-source projects, and the appeal of setting up a private Cloud, as opposed to using a commercial one, is that the administrator can have more control over the system. They support standard API interfaces (i.e., front-end that uses a subset of the EC2 interface), and they are often one of these customizable components. The last one is about the degree of transparency in the user interface. One of the main shared opinions in the commercial Cloud setting is the black-box nature of the system. The individual user, is not aware where, physically, his VMs are running. In a more customizable open-source setting, however, opportunities exist for a greater degree of explicit management with regard to the underlying configuration of physical machine and the location of the VMs. We remark that the authors of such a work have not recognized any features suitable for the cross Cloud management.

Nimbus [12] is an open source toolkit that allows to turn a set of computing resources into an Iaas Cloud. It was conceived from designers originally coming from the GRID world. Nimbus comes with a component called workspace-control, installed on each node, used to start, stop and pause VMs, implements VM image reconstruction and management, securely connects the VMs to the network, and delivers contextualization. Nimbus's workspace-control tools work with Xen and KVM but only the Xen version is distributed. Nimbus provides interfaces to VM management functions based on the WSRF set of protocols. There is also an alternative implementation exploiting Amazon EC2 WSDL. Its Federation system exploits the GRID-like existing functionalities. It leverages Virtual Organization (VOs) of GRID for controlling the access on virtual resources.

Eucalyptus [13] is an open-source Cloud-computing framework that uses the computational and storage infrastructures commonly available at academic research groups to provide a platform that is modular and open to experimental instrumentation and study. Eucalyptus addresses several crucial Cloud computing questions, including VM instance scheduling, Cloud computing administrative interfaces, construction of virtual networks, definition and execution of service level agreements (Cloud/user and Cloud/Cloud), and Cloud computing user interfaces. Not far past Eucalyptus was adopted as Virtualization Manager in the Ubuntu Core, but recently there is not longer support (Canonical switches to OpenStack for Ubuntu Linux Cloud [14]). The federation is out of the scope for them.

OpenNebula [15] is a virtualization tool to manage virtual infrastructures in a data-center or cluster, which is usually referred as private Cloud. Only the more recent versions of OpenNebula are trying to supports Hybrid Cloud to combine local infrastructure with public Cloud-based infrastructure, enabling highly scalable hosting environments. OpenNebula also supports Public Clouds by providing Cloud interfaces to expose its functionalities for virtual machine, storage and network management. The middleware tries to manage the federated resources but, considering the approach they use for interacting with physical servers (SSH remote shell commands), it is quite hard to accomplish real federation achievements. OpenNebula is mainly aimed at interoperability through OCCI interface.

A separated analysis has to be faced with the OpenStack [16] middleware because it operates in the direction of an open middleware for Clouds. The National Aeronautics and Space Administration (NASA) leads the project aiming to allow any organization to create and offer Cloud computing capabilities using open source software running on standard hardware. Openstack has three sub-projects that is OpenStack Compute, OpenStack Object Store and OpenStack Imaging Service. In particular OpenStack Compute is a software for automatically creating and managing large groups of virtual private servers. Open-Stack Storage is a software for creating redundant, scalable object storage using clusters of commodity servers to store terabytes or even petabytes of data. It adopts the Shared Nothing (SN), an architectural philosophy in which the platform is fully distributed and each node is independent and self-sufficient, and there is no single point of contention across the system. OpenStack Image Service is necessary for discovering, registering, and retrieving virtual machine images. The federation is not addressed at all in Openstack, the concepts Shared Nothing guarantees a high level of scalability and reliability, but at the same time the federation needs to be accomplished out of the architecture, at least as a *Federation Broker* that solves some of the federation issues. The SSO is only aimed at the dashboard web access, that is for the end-user.

III. The CLEVER IaaS Cloud

A. Overview

The CLEVER middleware is based on the architecture schema depicted in Figure 1, which shows a cluster of n nodes (also an interconnection of clusters could be analyzed) each containing a host level management module (Host Manager). A single node may also include a cluster level management module (Cluster Manager). All the entities interact exchanging information by mean of the Communication System based on the XMPP. The set of data necessary to enable the middleware functioning is stored within a specific Database deployed in a distributed fashion.

Figure 1 shows the main components of the CLEVER architecture, which can be split into two logical categories: the software agents (typical of the architecture itself) and the tools they exploit. To the former set belong both the Host Manager and the Cluster Manager:

- The Host manager (HM) performs the operations needed to monitor the physical resources and the instantiated VMs; moreover, it runs the VMs on the physical hosts (downloading the VM image) and performs the migration of VMs (more precisely, it performs the low

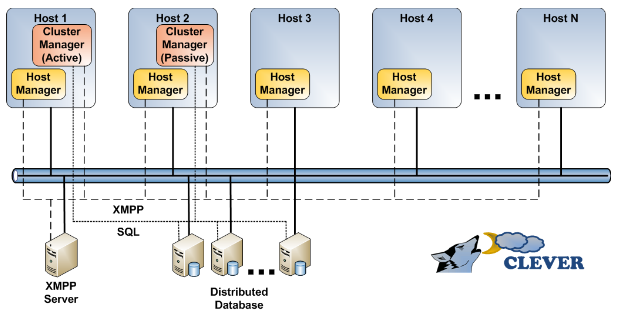

Figure 1. CLEVER architecture.

level aspects of this operation). To carry out these functions it must communicate with the hypervisor, hosts' OS and distributed file-system on which the VM images are stored. This interaction must be performed using a plug-ins paradigm.

- The Cluster Manager (CM) acts as an interface between the clients (software entities, which can exploit the Cloud) and the HM agents. CM receives commands from the clients, performs operations on the HM agents (or on the database) and finally sends information to the clients. It also performs the management of VM images (uploading, discovering, etc.) and the monitoring of the overall state of the cluster (resource usage, VMs state, etc.). At least one CM has to be deployed on each cluster but, in order to ensure higher fault tolerance, many of them should exist. A master CM will exist in active state while the other ones will remain in a monitoring state.

Regarding the tools such middleware components exploit, we can identify the Distributed Database and the XMPP Server.

B. *Internal/External Communication*

The main CLEVER entities, as already stated, are the Cluster Manager and the Host Manager modules, which include several sub-components, each designed to perform a specific task. In order to ensure as much as possible the middleware modularity, these sub-components are mapped on different processes within the Operating System of the same host, and communicate each other exchanging messages. CLEVER has been designed for supporting two different types of communication: intra-module (internal) communication and inter-module (external) communication.

1) Intra-module (Internal Communication): The intra-module communication involves sub-components of the same module. Since they essentially are separated processes, a specific Inter Process Communication (IPC) has to be employed for allowing their interaction. In order to guarantee the maximum flexibility, the communication has been designed employing two different modules: a low level one implementing the IPC, and an high-level one instead acting as interface with the CLEVER components, which allows access to the services they expose.

For implementing the communication mechanism, each module virtually exchanges messages (horizontally) with the corresponding peer exploiting a specific protocol (as the horizontal arrows indicate in Figure). However, the real message flow is the one indicated by the vertical arrows: when the Component Communication Module (CCM) of the Component A aims to send a message to its peer on a different Component B, it will exploit the services offered by the underlying IPC module. Obviously, in order to correctly communicate, the CCM must be aware of the interface by means of these services are accessible. If all the IPC were designed according to the same interface, the CCM will be able to interact with them regardless both their technology and implementation.

Looking into the above mentioned mechanism, when the Component A needs to access a service made available from the Component B, it performs a request through its CCM. This latter creates a message which describes the request, then formats the message according to the selected communication protocol and sends it to its peer on the Component B by means of the underlying IPC module. This latter in fact, once received the message, forwards it to its peer using a specific container and a specific protocol. The IPC module on the Component B, after that such a container is received, extracts the encapsulated message and forwards it to the overlying CCM. This latter interprets the request and starts the execution of the associated operation instead of the Component A.

2) Inter-module (External Communication): When two different hosts have to interact each other, the inter-module communication has to be exploited. The typical use cases refer to:

- Communication between CM and HM for exchanging information on the cluster state and sending specific commands;
- Communication between the administrators and CM using the ad-hoc client interface.

As previously discussed, in order to implement the inter-module communication mechanism, an XMPP server must exist within the CLEVER domain and all its entities must be connected to the same XMPP room.

When a message has to be transmitted from the CM to an HM, as represented in Figure 2, it is formatted and then sent using the XMPP. Once received, the message is checked from the HM, for verifying if the requested operation can be performed.

As the figure shows, two different situations could lay before: if the request can be handled, it is performed sending eventually an answer to the CM (if a return value is expected), otherwise an error message will be sent specifying an error code. The "Execution Operation" is a sub-activity

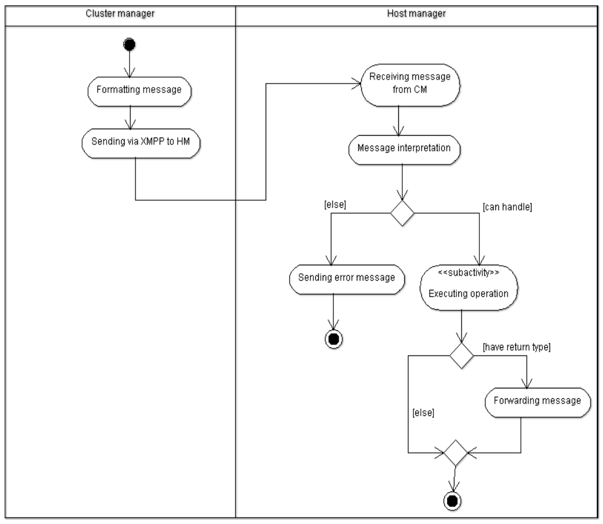

Figure 2. Activity diagram of the external communication.

whose description is pointed out in Figure 3. When the sub-activity is performed, if any return value is expected the procedure terminates, else this value has to be forwarded to the CM in the same way has been done previously with the request.

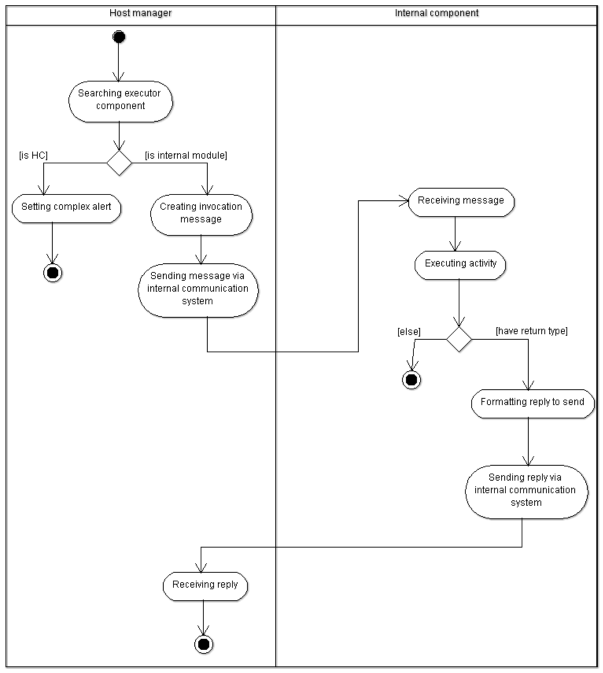

Figure 3. Activity Diagram of the sub-activity Executing Operation.

The sequence of steps involved in the sub-activity is represented in Figure 3. If the operation that has to be executed involves a component different from the Host Coordinator, the already described intra-module communication has to be employed. Once the selected component receives the message using this mechanism, if no problem occurs, the associated activity will be performed, else an error will be generated. If the operation is executed correctly and a return value has to be generated, the component will be responsible of generating the response message which will be forwarded to the HM, and thus, to the CM.

C. Federation Features

CLEVER has been designed with an eye toward federation. In fact, the choice of using XMPP for the CLEVER module communication (i.e., external communication XMPP room) has been made thinking about the possibility to support in the future also interdomain communication between different CLEVER administrative domains. Federation allows Clouds to “lend” and “borrow” computing and storage resources to/from other Clouds. In the case of CLEVER, this means that a CM of an administrative domain is able to control one or more HMs belonging other administrative domains. For example, if a CLEVER domain A runs out of resources of its own HMs, it can establish a federation with a CLEVER domain B, in order to allow the CM of the domain A to use one or more HMs of the domain B. This enables the CM of domain A to allocate VMs both in its own HMs and in the rented HMs of domain B. In this way, on one hand the CLEVER Cloud of domain A can continue to allocate services for its clients (e.g., IT companies, organization, desktop end-users, etcetera), whereas on the other hand the CLEVER Cloud of domain A earns money from the CLEVER Cloud of domain B for the renting of its HMs.

As anyone may run its own XMPP server on its own domain, it is the interconnection among these servers that exploits the interdomain communication. Usually, every user on the XMPP network has an unique Jabber ID (JID). To avoid requiring a central server to maintain a list of IDs, the JID is structured similarly to an e-mail address with an user name and a domain name for the server where that user resides, separated by an at sign (@). For example, considering the CLEVER scenario, a CM could be identified by a JID bach@domainB.net, whereas a HM could be identified by a JID liszt@domainA.net: bach and liszt respectively represent the host names of the CM and the HM, instead domainB.net and domainA.net represent respectively the domains of the Cloud which “borrows” its HMs and of the Cloud which “lends” HMs. Let us suppose that bach@domainB.net wants to communicate with liszt@domainA.net, bach and liszt, each respectively, have accounts on domainB.net and domain A XMPP servers.

The idea of CLEVER federation is straightforward by means of the built-in XMPP features. Figure 4 depicts an example of interdomain communication between two CLEVER administrative domains for the renting of two HMs from a domain A to domain B.

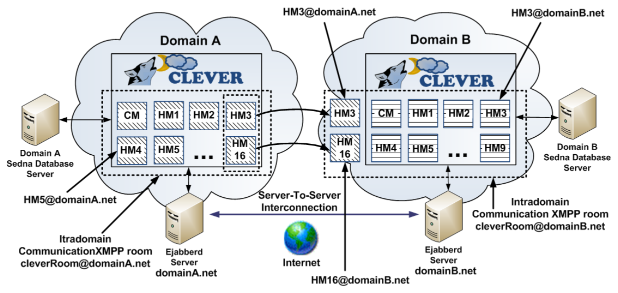
HM3@domainA.net
HM3@domainB.net
Domain A
Domain B
CLEVER
CLEVER
CM
HM1
HM2
HM3
HM4
HM5
...
HM 16
HM3
HM 16
CM
HM1
HM2
HM3
HM4
HM5
...
HM9
Domain A Sedna Database Server
Domain B Sedna Database Server
HM5@domainA.net
Server-To-Server Interconnection
Intradomain Communication XMPP room cleverRoom@domainB.net
Itradomain CommunicationXMPP room cleverRoom@domainA.net
Ejabberd Server domainA.net
Internet
Ejabberd Server domainB.net
HM16@domainB.net

Figure 4. Example of CLEVER in horizontal federation.

Considering the aforementioned domains, i.e., domainA.net and domainB.net, in scenarios without federation, they respectively include different XMPP rooms for intradomain communication (i.e., cleverRoom@domainA.net and cleverRoom@domainB.net) on which a single CM, responsible for the administration of the domain, communicates with several HMs, typically placed within the physical cluster of the CLEVER domain. Considering a federation scenario between the two domains, if the CM the domainB.net domain needs of external resources, after a priori agreements, it can invite within its cleverRoom@domainB.net room one or more HMs of the domainA.net domain. For example, as depicted in Figure 4, the CLEVER Cloud of domainB.net rents from the CLEVER Cloud of domainA.net, HM6 and HM16. Thus, the two rented HMs will be physically placed in domainA.net, but they will be logically included in domainB.net. As previously stated, in order to accomplish such a task a trust relationship between the domainA.net and the domainB.net XMPP servers has to be established in order to enable a Server-to-Server communication allowing to HMs of domain A to join the external communication XMPP room of domain B.

IV. Authentication Issues in CLEVER Federation

Federation between CLEVER Clouds implies the establishment of a secure inter-domain communication between their own XMPP servers. This raises several issues regarding the management of authentication between the XMPP servers of different CLEVER Clouds. In this section, after a discussion of the authentication mechanisms supported by XMPP for the establishment of a server-to-server federation, we describe the authentication issues in a scalable scenario of federated CLEVER Clouds, proposing a solution based on the IdP/SP model.

A. Concerns about XMPP Server-to-Server Federation

Considering that the communication in each CLEVER Cloud is achieved through XMPP or Jabber messages by means of an Ejabberd server, the federation establishment between two or more CLEVER Clouds implies a secure inter-domain communication between their respective Ejabberd servers. In fact, in the XMPP terminology, the term "federation" is commonly used to describe communication between two servers.

The public-subscribe technology is reemerging for enabling real-time communication within Cloud infrastructure, nevertheless its major protocol XMPP is somewhat dated from the point of view of security.

In order to enable federation between servers, it is needed to carry out a strong security to ensure both authentication and confidentiality thanks to encryption. According to the IETF 6120, compliant implementations of servers should support Dialback or SASL EXTERNAL protocol for authentication and the TLS protocol for encryption.

The basic idea behind Server Dialback [17] is that a receiving server does not accept XMPP traffic from a sending server until it has (i) "called back" the authoritative server for the domain asserted by the sending server and (ii) verified that the sending server is truly authorized to generate XMPP traffic for that domain. The basic flow of events in Server Dialback consists of the following four steps:

1) The Originating Server generates a dialback key and sends that value over its XML stream with the Receiving Server. (If the Originating Server does not yet have an XML stream to the Receiving Server, it will first need to perform a DNS lookup on the Target Domain and thus discover the Receiving Server, open a TCP connection to the discovered IP address and port, and establish an XML stream with the Receiving Server.)
2) Instead of immediately accepting XML stanzas on the connection from the Originating Server, the Receiving Server sends the same dialback key over its XML stream with the Authoritative Server for verification. (If the Receiving Server does not yet have an XML stream to the Authoritative Server, it will first need to perform a DNS lookup on the Sender Domain and thus discover the Authoritative Server, open a TCP connection to the discovered IP address and port, and establish an XML stream with the Authoritative Server).
3) The Authoritative Server informs the Receiving Server whether the key is valid or invalid.
4) The Receiving Server informs the Originating Server whether its identity has been verified or not.

SASL is a framework for providing authentication and data security services in connection-oriented protocols via replaceable mechanisms. It provides a structured interface between protocols and mechanisms. The resulting framework allows new protocols to reuse existing mechanisms and allows old protocols to make use of new mechanisms. SASL is used in various application protocols (e.g., XMPP, IMAP, LDAP, SMTP, POP, etc.) and support many mechanisms including:

- **PLAIN**, a simple clear text password mechanism. PLAIN obsoleted the LOGIN mechanism.

- **SKEY**, an S/KEY mechanism.
- **CRAM-MD5**, a simple challenge-response scheme based on HMAC-MD5.
- **DIGEST-MD5**, HTTP Digest compatible challenge-response scheme based upon MD5. DIGEST-MD5 offers a data security layer.
- **GSSAPI**, for Kerberos V5 authentication via the GSS-API. GSSAPI offers a data-security layer.
- **GateKeeper**, a challenge-response mechanism developed by Microsoft for MSN Chat

At the time of writing of the IETF 6120, in March 2011, most server implementations still use the Dialback protocol to provide weak identity verification instead of using SASL to provide strong authentication, especially in cases where SASL negotiation would not result in strong authentication anyway (e.g., because TLS negotiation was not mandated by the peer server, or because the PKIX certificate presented by the peer server during TLS negotiation is self-signed and has not been previously accepted). The solutions is to offer a significantly stronger level of security through SASL and TLS.

B. SASL and SAML for Secure CLEVER Federation

In a scalable scenario of federation each CLEVER Cloud can require to frequently establish/break partnerships with other CLEVER Clouds. This implies that each Cloud should manage a huge number of credentials in order to authenticate itself in other Clouds. In a federated CLEVER environment, this means that the XMPP server of the Cloud requiring federation has to be authenticated by the XMPP server of the Cloud accepting the federation request. If we consider thousand of Clouds, each Cloud should manage one credential for accessing to each federated Cloud. This is problem is commonly known as Single-Sign-One (SSO), i.e., considering an inter-domain environment, performing the authentication once, gaining the access to the resources supplied by different Service Provider, each one belonging to a specific domain. A model addressing the SSO problem is the Identity Provider/Service Provider Model (IdP/SP). Typically, a client who wants to access to the resources provided by a SP, perform the authentication once on the IdP (asserting party), which asserts to the SP (relaying party) the validity of the authentication of the client. Considering many SPs relaying on the IdP if the client wants to access another SP, as this latter will be trusted with the IdP, no further authentication will be required. This model is widely known on the Web with the term "Web Browser SSO", in which the client is commonly an user who perform an authentication fill in an HTML form with his user name and password. Nowadays, the major standard implementing defining the IdP/SP model is the Security Assertion Markup Language (SAML) [8], developed by OASIS.

The scenario of CLEVER federation is quite similar. In this case, the client who wants to perform the authentication is the XMPP server of the CLEVER Cloud requiring federation, instead the role of the SP is played by the XMPP server of the Cloud accepting the federation request. As the XMPP server support authentication through SASL a concern raises: the RFC 4422 does not support any security mechanism implementing the IdP/SP model.

Therefore, in order to achieve such a scenario, we followed the Internet-Draft entitled "A SASL Mechanism for SAML", defined by CISCO TF-Mobility Vienna, describing the applicability and integration between the two protocols for non-HTTP use cases. According to such a draft, the authentication should occur as follows:

1) The server MAY advertise the SAML20 capability.
2) The client initiates a SASL authentication with SAML20
3) The server sends the client one of two responses:
    a) a redirect to an IdP discovery service; or
    b) a redirect to the IdP with a complete authentication request.
4) In either case, the client MUST send an empty response.
5) The SASL client hands the redirect to either a browser or an appropriate handler (either external or internal to the client),and the SAML authentication proceeds externally and opaquely from the SASL process.
6) The SASL Server indicates success or failure, along with an optional list of attributes

In this way, thanks to SASL and SAML, for each CLEVER Cloud it is possible to perform the authentication once gaining the access to all the other Clouds relying on the IdP, thence, lending and/or borrowing HMs according to agreements.

V. Secure CLEVER Interdomain Communication Through Shibboleth Federation

In the previous section, we have analyzed different technologies able to address authentication issues in distributed environments, where users need to prove their identities. As we introduced earlier, some specific scenarios, such as Cloud Federation, may require that systems belonging to different administrative domains interact each other to cooperate. In this section, we try to extend the mechanisms regarding authentication in distributed environment toward Cloud systems, proposing our idea for implementing Single Sign On among different XMPP servers, in order to grant either scalability and flexibility while the authentication process is accomplished.

In a Cloud Federated scenario, where each Cloud refers to CLEVER as Virtual Infrastructure Manager, and the communication among its entities is thus based on XMPP, the most convenient and easy way to build a Cloud federation should rely on the employment of the federation features made available by the XMPP protocol itself. This latter assumes a XMPP server can be configured for accepting external connections from other servers for creating server-to-server interactions (server federation).

According to the XMPP specifications, this mechanism is quite easy to implement and the result will be the ability for two XMPP servers in different domains to exchange XML stanzas. There are different levels of federation:

- Permissive Federation, a server accepts a connection from any other peer on the network, even without verifying the identity of the peer based on DNS lookups.
- Verified Federation, a server accepts a connection from a peer only after the identity of the peer has been weakly verified via Server Dialback, based on information obtained via the Domain Name System (DNS) and verification keys exchanged in-band over XMPP.
- Encrypted Federation, a server accepts a connection from a peer only if the peer supports Transport Layer Security (TLS) and the client authenticates itself using a SASL mechanisms.

On one hand, Permissive and Verified Federation are the simplest federation approaches: as discussed in the previous Section, they lack some security aspects since they are not based on any password exchange procedure and, in order to implement domain filtering (in the second case), a list of allowed sites has to be compiled preemptively. On the other hand, the Encrypted Federation level relies on a more secure way to perform the authentication, based on challenge-response authentication protocols relaying on passphrase.

This standard authentication mechanisms are enough when you want to enable the communication among a limited endpoint number but, in a scenario where several XMPP servers might exist, it could be a difficult task to statically pre-configure the binding among all the involved entities and manage credentials for authenticating a given server to each other. Our idea aims to address these issues and propose the integration of a new SASL security mechanism for allowing a more scalable management of the authentication process exploiting the well-known concept of SSO. The integration we are talking about refers to the use of SAML 2.0.

In ordeer to implement the above mentioned scenario, we arranged a distributed Cloud environment composed of two CLEVER sites relying on the Ejabberd XMPP server [9] to allow communication within each domain. Furthermore, in order to verify the server-to-server federation we configured each server to listen for incoming connection on a given port. This task is usually accomplished by an Ejabberd module that manages incoming and outcoming connections from/to external servers. According to the XMPP core specification, this module is able to establish server federation according to the three different levels pointed out above. In our work we considered more specifically the Encrypted Federation case and we have modified the Ejabberd module performing SASL to add in the list of the supported security mechanism also SAML 2.0. This latter has been introduced relying on an external software module based on Shibboleth named Authentication Agent (AA).

The Authentication Agent acts as user when it is contacted from the Source Ejabberd Server for starting the Federation, whereas represents the Relying Party when it is contacted from the Destination Ejabberd Server.

In the following, we present the sequence of steps performed by two servers (for simplicity Source Server and Destination Server) that aim to build the federation. As Figure 5 depicts, the involved actors in the process are the s2s_Manager(s) of both the Ejabberd servers, the two authentication agents acting as User and Relying Party (User, the one interacting with the Source Server; Relying Party the one interacting with the Destination Server) and the Identity Provider (also implemented using Shibboleth).

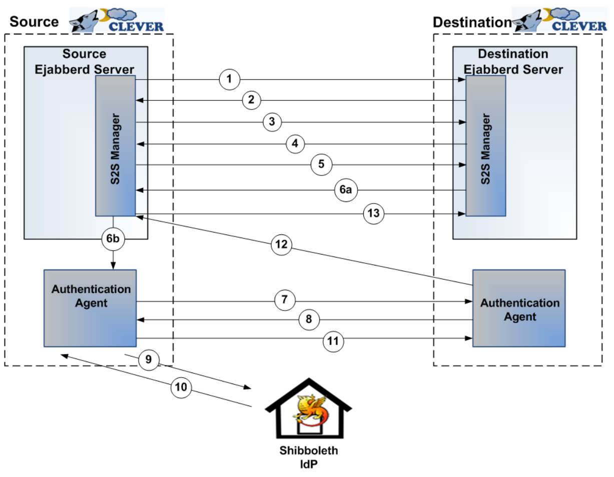

Figure 5. Step performed by two XMPP servers aiming to build Federation: the authentication process is executed using SAML 2.0 as external SASL mechanism

- Step 1: s2s_Manager of Source Server initiates stream to the s2s_Manager of the Destination server.
- Step 2: s2s_Manager of the Destination Server responds with a stream tag sent to the s2s_Manager of the Source Server.
- Step 3: s2s_Manager of the Destination Server informs the s2s_Manager of the Source Server of available authentication mechanisms.
- Step 4: s2s_Manager of the Source Server selects SAML as an authentication mechanism.
- Step 5: s2s_Manager of Destination Server sends a BASE64 encoded challenge to the s2s_Manager of the Source Server in the form of an HTTP Redirect to the Destination AA (acting as Relying Party).
- Step 6: a) s2s_Manager of Source Server sends a BASE64 encoded empty response to the challenge and b) forward to the Source AA the URL of the Relying Party.

- Step 7: The Source AA (User) engages the SAML authentication flow (external to SASL) contacting the Destination AA (Relying Party).
- Step 8: Destination AA redirect Source AA to the IdP.
- Step 9: Source AA contacts IdP and performs Authentication
- Step 10: IdP responds with Authentication Assertion
- Step 11: Source AA contacts Destination AA for gaining access to the resource.
- Step 12: Destination AA contacts the s2s_Manager of the Destination Server informing it about the authentication result.
- Step 13: if the authentication is successful the s2s_Manager of the Source Server initiates a new stream to the s2s_Manager of Destination Server.

The advantage of performing the authentication among servers in such a way mainly consists in the higher security level achieved than the traditional Dialback/SASL mechanisms and in the possibility of exploiting the SSO authentication. Looking at Figure 5, after that the federation has been achieved with the depicted server, if the same Source Cloud aims to perform server-to-server federation with a new XMPP server that relies on the same IdP as trusted third-party, such a process would be straightforward. Since the Source Server already has an established security context with the IdP, once the SASL process starts and the SAML mechanism is selected, no further authentication will be required.

VI. Conclusions and Remarks

In this paper, we discussed how to perform the authentication among CLEVER Clouds in order to establish federation. CLEVER is an IaaS Cloud middleware designed according to the public-subscribe technology and implementing the XMPP protocol. The federation establishment involves the federation among their own XMPP servers. Considering the current implementation of XMPP servers, server-to-server federation implies security issues due to authentication. In particular the SASL framework used in XMPP server does not provide any SSO authentication mechanism, a mandatory requirement for a scalable federated Cloud environment. In order to address this issue, in this work, we used an integration of SASL and SAML implementing a testbed including Ejabberd as XMPP server and Shibboleth as SAML implementation. Experiments have proved that such a solution can be a valid approach for a federation-enabled Cloud infrastructure using a public-subscribe technology, such as CLEVER.

References

[1] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Munoz, and G. Toffetti, "Reservoir - when one cloud is not enough," *Computer*, vol. 44, pp. 44–51, 2011.

[2] B. Sotomayor, R. Montero, I. Llorente, and I. Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds," *Internet Computing, IEEE*, vol. 13, pp. 14–22, Sept.-Oct. 2009.

[3] National Institute of Science and Technology. Standards Acceleration to Jumpstart Adoption of Cloud Computing; http://csrc.nist.gov/groups/SNS/cloud-computing/ July 2011.

[4] F. Tusa, M. Paone, M. Villari, and A. Puliafito., "CLEVER: A CLoud-Enabled Virtual EnviRonment," in *15th IEEE Symposium on Computers and CommunicationsS Computing and Communications, 2010. ISCC '10. Riccione*, June 2010.

[5] Extensible Messaging and Presence Protocol (XMPP), http://xmpp.org/, Jan 2012.

[6] RFC 6120, Extensible Messaging and Presence Protocol (XMPP): Core, http://tools.ietf.org/rfc/rfc6120.

[7] RFC 4422, Simple Authentication and Security Layer (SASL), http://www.ietf.org/rfc/rfc4422.

[8] SAML V2.0 Technical Overview, OASIS, http://www.oasis-open.org/specs/index.php#saml, Jan 2012.

[9] Ejabberd, the Erlang Jabber/XMPP daemon: http://www.ejabberd.im/, Jan 2012.

[10] The Shibboleth system standards, Available: http://shibboleth. internet2.edu/, Jan 2012.

[11] P. Sempolinski and D. Thain, "A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus," in *The 2nd IEEE International Conference on Cloud Computing Technology and Science*, July 2010.

[12] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the Use of Cloud Computing for Scientific Workflows," in *SWBES 2008, Indianapolis*, December 2008.

[13] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," in *IEEE/ACM CCGRID*, pp. 124–131, May 2009.

[14] Canonical switches to OpenStack for Ubuntu Linux cloud. http://www.zdnet.com/blog/open-source/canonical-switches-to-openstack-for-ubuntu-linux-cloud/8875, Jan 2012.

[15] B. Sotomayor, R. Montero, I. Llorente, and I. Foster, "Resource Leasing and the Art of Suspending Virtual Machines," in *HPCC*, pp. 59–68, June 2009.

[16] OpenStack: Open source software for building private and public clouds. http://www.openstack.org/, Jan 2012.

[17] XEP-0220: Server Dialback, http://xmpp.org/extensions/xep-0220.html.