EAI Endorsed Transactions
on Smart Cities

Research Article EAI.EU

C-ABAC: An ABAC based Model for Collaboration in Multi-tenant Environment

Mohamed Amine Madani[1,*], Mohammed Erradi[1], Yahya Benkaouz[2]

[1]Networking and Distributed Systems Research Group, ITM Team, ENSIAS, Mohammed V University in Rabat, Morocco
[2]Conception and Systems Laboratory, FSR, Mohammed V University in Rabat, Morocco

Abstract

Collaborative systems allow a group of users to collaborate through distributed platforms in order to perform a common task. Collaborators usually use cloud-based solutions to outsource their data and to benefit from the cloud capabilities. Ensuring access control in a cloud-based collaborative session is an important problem that should be addressed, especially in a multi-tenant configuration. In this paper, we present C-ABAC, a Collaboration Attributes Based Access Control model that ensures access control in multi-tenant cloud environments. C-ABAC supports the workflow concept, preserves the tenants autonomy in defining their local policies and preserves the confidentiality of the object attributes. The implementation of C-ABAC in the SwiftStack environment demonstrates the feasibility of the suggested model.

Received on 15 December 2017; accepted on 18 April 2018; published on 26 June 2018

Keywords: ABAC model; Tasks; Collaborative session; Access control.

doi:10.4108/eai.26-6-2018.154831

1. Introduction

Nowadays, multiple organizations collaborate by performing common tasks in order to reach a common goal. Such collaborations optimize the usage of the distributed resources of the collaborators, hence, the productivity and the benefit improvements. In this context, collaborative applications bring new solutions and technologies to enable a group of users to communicate, cooperate and collaborate through distributed platforms to perform common tasks.

Most organizations rely on cloud-based solutions to outsource their IT infrastructure such as compute, network and data storage in a cloud service provider (CSP). This provides remote access to software and hardware services via Internet. In order to ensure the confidentiality and the privacy of these services, the cloud service provider segregates the data and customers services into multiple tenants. Each tenant is assigned to an organization or to a person that uses a given cloud service.

During collaborations, the cloud tenants need to access and use the information shared by other collaborating tenants. This information often contains sensitive data. It is meant to be shared only during specific collaborative sessions [5]. This arises the access control issue [4]: The tenants need strong access control model supporting cross tenants access and collaboration. Moreover, users may intervene dynamically without a prior knowledge of which user will request an access to a given object. In this direction, designing a fine-grained access control model is mandatory [5].

Note that a collaboration might be seen as a set of tasks and workflows. Each task is performed by a given tenant and a tenant may achieve one or more tasks. A task might be active (i.e. a part of a workflow) or passive (i.e. does not belong to a workflow). On the other hand, access control models for collaborations in multi-tenant environments might be classified into two categories: Centralized and decentralized (peer to peer) access control models. In centralized approaches, the access enforcement and decisions are taken in a

★Please ensure that you use the most up to date class file, available from EAI at http://doc.eai.eu/publications/transactions/latex/

*Corresponding author. Email: amine.madani@um5s.net.ma

specific central point. These models enable granting and revoking permissions while a task is running. In decentralized approaches, each tenant is responsible of its own access control policy. In this respect, tenants are loosely coupled. Such approaches could support the following requirements:

- Autonomy and independence: Each tenant maintains control over its resources. Each tenant defines its local access control policy and respects the global access control policy.
- Confidentiality: Each tenant is be able to maintain the confidentiality and the privacy of its local policy and its own data.

In this paper, and based on the fine-grained access control model "ABAC: Attribute based access control", we propose "C-ABAC: Collaboration ABAC". C-ABAC is especially designed for collaborations in multi-tenants environments. C-ABAC model overcomes the limitations of the classical access control models that are based on RBAC model. It supports an access cross-tenant in which a tenant could use shared ressources on the cloud while preserving access control policies. The C-ABAC model is a centralized model that allows the collaborating tenants to specify a global policy (authorizations) in a specific central point. C-ABAC supports the task and the workflow concepts. It is scalable and preserves the autonomy of each tenant in defining their policies. In addition, it preserves the confidentiality of the object, resource and environment attributes that are used in the access decision process.

This paper is organized as follows: Section 2 presents the background of this work, in which, the concepts of Cloud-based collaborative applications and the ABAC model are explained. The related work are presented in Section 3. Section 4 describes the suggested C-ABAC model. Section 5 presents the implemented architecture, the enforcement model and discusses the evaluation results. Finally, we conclude in Section 6.

2. Background

This section aims to present the necessary background of this work. It mainly focus on the presentation of the concept of Cloud based collaborative application. Then, it presents the attribute based access control model.

Cloud based collaborative applications.

Collaborative applications are among the services that can be provided by the cloud computing. They enable collaboration among users from the same or different tenants of a given cloud provider [2, 3]. During collaborations, the participants need to access and use resources held by other collaborating users. These resources often contain sensitive data. They are meant to be shared only during specific collaborative session [5]. The collaborative session is an abstract entity, comprising a set of users, called members of the session. These members play either the same role or different roles. They might have concurrent access to shared objects in the collaborative session depending on the access control policy.

Case study: a collaborative application for telemedicine

In this study, we consider the telemedicine scenario shown in Figure 1. In this real use case, the School Hospital *(SH)*, the Emergency Medical Services *(EMS)*, and the Home Hospital *(HH)* are three collaborating issuers sharing a common private cloud service. The cloud service provides storage services for the Home Hospital issuer, and for the three SH's departments: *neurology*, *radiology* and *cardiology*, as segregated tenants.

This private cloud provides a service of collaborative sessions for the Emergency medical services *(EMS)*. This service allows a group of users, from different tenants, to collaborate in order to observe and treat a patient admitted in the Home Hospital *(HH)* emergency. In this scenario, we have a collaborative session *CS1* of a *telemedicine type*. The members of this session are:

- User1: *neurologist* in the tenant *neuro* of the issuer SH;
- User2: *cardiologist* in the tenant *cardio* of the issuer SH;
- User3: *radiologist* in the tenant *radio* of the issuer SH;
- User4: *doctor_EMS* (Emergency doctor) in the tenant *emr* of the issuer EMS;
- User5: *doctor_HH* in the tenant *storage* of the tenant HH.

ABAC Model.

ABAC is an adaptive and a flexible fine-grained access control model. The core components of ABAC model [9] are:

- U, O and E represent finite sets of existing users, objects and environments respectively.
- $A = \{create, read, update, delete\}$ is a finite set of actions.
- $UATT$, $OATT$ and $EATT$ represent finite sets of user, object and environment attribute functions respectively.
- For each $att \in \{UATT \cup OATT \cup EATT\}$, $range(att)$ represents the attribute's range, which is a finite set of atomic values.

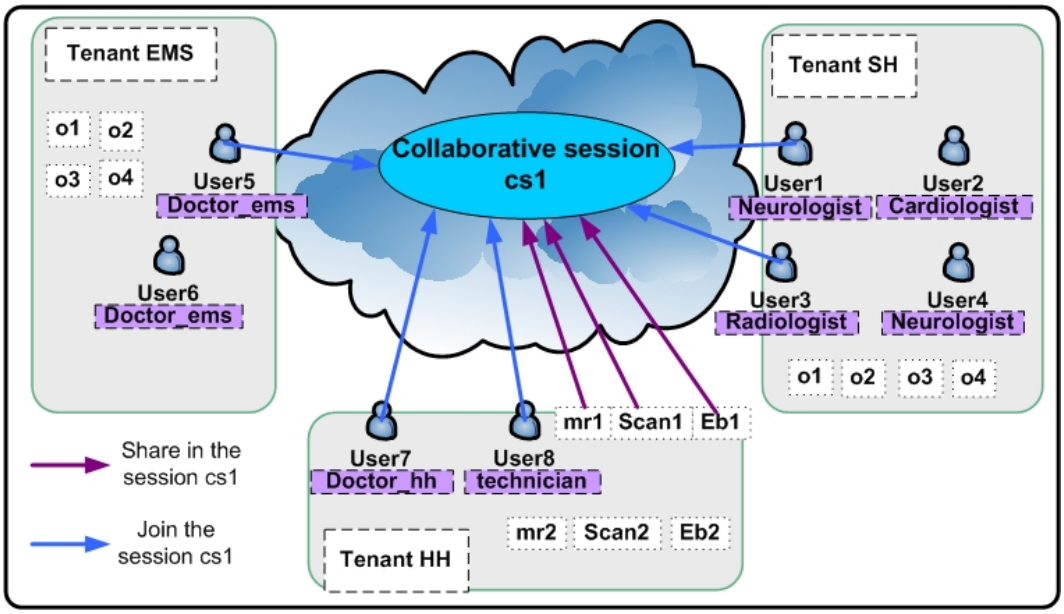

Figure 1. A collaborative session in cloud environment

- $attType : UATT \cup OATT \cup EATT \rightarrow \{set, atomic\}$, specifies attributes as set or atomic values.
- Each attribute function maps elements in U to an atomic value or a set
  - $\forall ua \subseteq UATT.\ ua : U \rightarrow Range(ua)\ if\ attType(ua) = atomic$
  - $\forall ua \subseteq UATT.\ ua : U \rightarrow 2^{Range(ua)}\ if\ attType(ua) = set$
- Each attribute function maps elements in O to an atomic value or a set
  - $\forall oa \subseteq OATT.oa : O \rightarrow Range(oa)\ if\ attType(oa) = atomic$
  - $\forall oa \subseteq OATT.oa : O \rightarrow 2^{Range(oa)}\ if\ attType(oa) = set$
- Each attribute function maps elements in E to an atomic value or a set
  - $\forall ea \subseteq EATT.ea : E \rightarrow Range(ea)\ if\ attType(ea) = atomic$
  - $\forall ea \subseteq EATT.ea : E \rightarrow 2^{Range(ea)}\ if\ attType(ea) = set$
- An authorization that decides on whether a user u can access an object o in a particular environment e for the action a, is a boolean function of u, o, and e attributes: Rule: $authorization_a(u, o, e) \rightarrow f(ATTR(u), ATTR(o), ATTR(e))$.

3. Related work

Several works have been in the literature to ensure access control in multiple environments. In the Task based access control [4] (TBAC), the permissions are granted according to the progress of several tasks. The TRBAC [18] model is constructed by adding the "Task" concept to the RBAC model. In TRBAC, the user has a relationship with permission through role and task. On the other hand, in the Team Access Control Model (TMAC) [6], the permissions are granted to each user through its role and the current activities of the team. These models enable fine-grained access control but they do not incorporate contextual parameters into security considerations and do not support collaboration in multi-tenants environments. Moreover, the notion of "Team" used in TMAC model is static. Therefore, this model does not support dynamic collaboration.

Other access control approaches have been suggested to secure resources in cloud environments [2, 3, 19–21]. Calero [2] suggests a multi-tenancy authorization system. This work is based on hierarchical role-base access control with a coarse-grained trust relation and path-based object hierarchies . Calero et al [2] assumes that each issuer may use several cloud services and could collaborate with other issuers. Tang [20] proposes a multi-tenancy authorization system (MTAS) model. This model is based on the RBAC model and the trust relations established between the cloud issuers in order to support collaboration between these issuers. The issuer that establishes the trust is called the *truster* and the one being trusted is called the *trustee*. The trustee

can authorize one of the trusters roles to access to a trustee resource.

The Multi-Tenant Role-Based Access Control (MT-RBAC) proposed by Tang et al in [3] is a model that provides fine-grained access control in collaborative cloud environments by using trust relations among tenants. In this work, trust relations among issuers were not considered (i.e. they distinguish between issuers and tenants). In MT-RBAC model, the truster exposes some trusters roles to the trustee. This trustee assigns their users to the trusters roles. Thus, the users can access to the trusters resources by activating the trusters roles.

In Collaborative Task Role-Based Access Control CTRBAC model [21], authors propose an approach to ensure access control to the shared resources in a collaborative session in multi-tenants environments. The suggested CTRBAC model is an extended version of RBAC in which new entities were added in order to support together the cross tenants access and the task concept. Nonetheless, in this model, a given tenant may use some roles owned by other tenants which will compromise the confidentiality requirement. Furthermore, this model is based on RBAC model which is not flexible enough to support a complex policy rules.

These models are based on a decentralized approach. It supports the following requirements: (1) Autonomy and independence: each local administrator maintains control over his system. Each organization defines its local access control policy, and respects the global access control policy. (2) Cross tenant access: Tenant uses some resources shared by other tenants. However, these models do not support task and scalability requirements, especially if we assume that the collaboration is a workflow composed of a set of tasks. Moreover, these approaches are based on Role based access control (RBAC) Model. Nevertheless, various limitations of RBAC have been recognized such as: flexibility and scalability.

In this direction, Attribute Based Access Control model (ABAC) is of a great interest. ABAC model [9, 10] overcomes the limitations of the classical access control models (i.e, ACL, MAC and RBAC). This model is adaptive and flexible. ABAC is more suitable to describe complex, fine-grained access control semantics, which is especially needed for collaborative environments.

There have been few works that used ABAC in multi-tenant environment. The multi-tenant attribute-based access control model (MT-ABAC)[11] presents model to enable collaboration between tenants in the cloud. This model is based on a decentralized approach and supports cross-tenant attribute assignment. However, this model does not support the task concept. In MT-ABAC Model, authors defined a trust relationship established between the truster tenant and the trustee tenant in order to support cross tenants access. In this relationship, the trustee is authorized to assign values for trustee's user attributes to truster's users. However, before assigning the users to the attributes, the trustee should know some informations about truster's users such as their jobs in the organization which will compromise the confidentiality requirement. Moreover, the trustee has the full control to assign the truster's users to the trustee's authorizations which will compromise the autonomy requirement.

Therefore, in this paper, we propose C-ABAC, a novel ABAC based model following a centralized approach. C-ABAC allows the collaborating tenants to specify the global policy in a specific central point. C-ABAC supports the concepts of task and workflow. It ensures the tenants autonomy and preserves the attributes confidentiality of the tenants objects.

4. C-ABAC: The Collaboration ABAC model

In this section, we present the suggested collaboration attributes based access control model: C-ABAC. In this section, we first define the notion of collaborative tenant. Then, we describe the business process for the collaboration. After that, we present the C-ABAC model definition. Finally, we show a use how C-ABAC might be used in the previously described Telemedicine use case.

4.1. A collaborative tenant

The collaborative tenant is the tenant responsible for ensuring the collaboration between multi tenants. It provides the collaboration as a service for the collaborating tenants. This collaborative tenant allows a group of users from different tenants to collaborate through distributed platforms in order to perform a common process. Note that each set of tenants that want to collaborate with each other should first create this collaborative tenant. Then, they should define in this collaborative tenant the collaboration process which is a workflow composed of a set of tasks, each task will be performed by a given tenant and a tenant may achieve one or more tasks. For instance as shown in figure 2, the tenants *SH*, *EMS* and *HH* are three collaborating tenants using the collaborative tenant that provides the collaboration as a service.

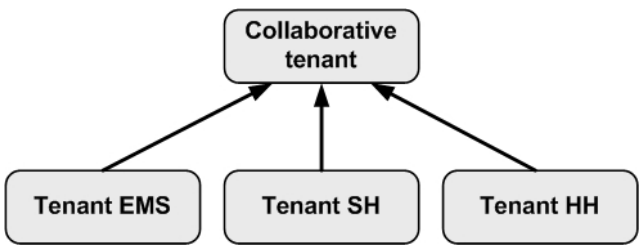

Figure 2. Collaboration As A Service (CAAS)

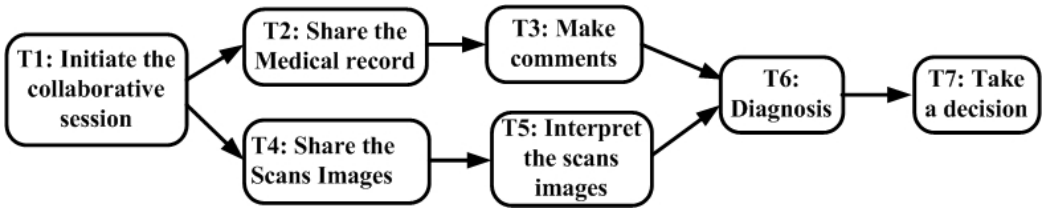

Figure 3. A collaboration workflow

4.2. A business process

In our approach, each collaboration is specified as a business process that is defined as a set of tasks that are connected to achieve a common goal. For example, figure 3 shows the diagnosis process related to the our use case. In this collaboration, each task will be performed by a given tenant using some resources shared by others tenants. As shown in the tasks assignment (figure 4), the tenant SH is responsible to achieve the task $T7$. Moreover, in order to perform this task, the tenant SH needs to access to the resources MR, $Scan$ and $video$ that are owned by the tenant HH.

In this section, we present our core C-ABAC model which is designed to be suitable for ensuring access control in collaborative multi-tenants environments. ABAC model has been defined in various ways in the literature, usually for some specific goals. In our approach, we add the tasks (T) entity in addition to the users and objects of core ABAC0. The task is a fundamental unit of business work or business activity. Tasks are assigned to tenants according to their roles in the collaboration.

The task is defined using the triple (task name, tenant that is responsible to achieve the task, Set of resources (owned by others) tenants used for achieving the task). For example as shown in figure 4, the tenant SH achieves the task $T7$ by using some resources owned by the tenant HH. In order to specify ABAC authorization while supporting collaboration and tasks requirements related to multi-tenants environments, we should use the notation illustrated in figure 5.

C-ABAC model introduces the task entity to the user and object entities of the ABAC model. In this model, each task is defined by a set of task attributes like: the task name, the workflow of the task, previous tasks and the collaborative session of the task. Moreover, in each task, the responsible of this task may have many authorizations. For example in the task $T7 : Take\_a\_decision$, a user from the tenant SH needs have four authorizations: (1) read the patient medical record mr1; (2) read the patient scan image scan1; (3) read the patient video file; (4) write the final decision.

In C-ABAC model, each authorization related to a given task is defined as shown in figure 2 by: (1) Set of task attributes related to this task; (2) Set of user attributes that represent the user who is responsible to achieve this task; (3) Set of object attributes related to the resource used in this task; (4) An action which is a specific operation on object. For each task, the user that is responsible of the task should have many permissions to accomplish this task. So each task is assigned to many permissions (ABAC Authorization).

A C-ABAC Authorizations are defined using task, user, and object attributes that are independent of one another. Moreover, each task attributes have the same value for all C-ABAC authorizations related to this task. Likewise, each user attributes have the same value for all authorizations related to one task the fact that we consider that the user who is responsible to achieve one task is authorized to perform all actions related to this task.

A C-ABAC model is composed of the three basic components: users (U), objects (O), and Tasks (T). In this model, each user has an attribute $UOwner$ which is a many-to-one function from users U to tenants TE. Moreover, the model requires each object to have an attribute $OOwner$ which is a many-to-one function from objects O to tenants TE. Further, each user attribute, each object attribute and each task attribute is also uniquely owned by a single tenant, depicted respectively by the many atomic-valued functions $UAOwner$, $OAOwner$ and $TAOwner$.

The crucial concept is that each tenant is responsible for assigning values to attributes that it owns. With isolated tenants, a user can have assigned values only for those attributes owned by the user's owning tenant. Actions are allowed operations in the system. These operations typically include create, read, update and delete. We use the terms actions and operations interchangeably. An action is applied to an object by a user.

In our approach, a global C-ABAC authorizations for a given task of the collaboration includes global attributes which will be defined at the collaborative tenant level. Such attributes are the task attributes, user and object attributes related to the collaboration (for instance, $MemberCS(u)$: member of the collaborative session; $SharedCS(o)$: shared in the collaborative session). On the other hand, in this authorization, the administrateur of the collaborative tenant uses the attributes related to the user who is responsible

Tasks	The Assigned Users	The Used Objects
T5: Interpret the scans images	Tenant SH	Scan→Tenant HH
T7: Take a decision	Tenant SH	MR → Tenant HH Scan→Tenant HH Video→Tenant HH

Figure 4. Tasks Assignment

Tasks	The Assigned Users	The Used Objects	Action
$TATT1(ti) = \{Vali\}$ $TATT2(ti) = \{Vali\}$ …… $TATTn(ti) = \{Vali\}$	$UATT1(u) = \{valj\}$ $UATT2(u) = \{valj\}$ …… $AssignedUser_{(ta,te)}(u) = 'True'$	$OATT1(o) = \{valk\}$ $OATT2(o) = \{valk\}$ …… $UsedObject_{(OType,a,te)}(o) = 'True'$	*Read / Write*

Figure 5. C-ABAC Authorizations

for executing this task. These user attributes will be defined by the task executor (the tenant assigned for executing this task) in a confidential, autonomous and independent way from other collaborating tenants and the collaborative tenant. For this purpose, we propose a new attribute function $AssignUser()$ which will be defined at the level of the local tenant. This attribute function is responsible on executing the task and will be used by the collaborative tenant at the level of the global autorization.

Moreover, in this authorization the administrator uses the attributes related to the object shared within the task. Similarly, these object attributes will be defined by the object owner in a confidential and independent way from the other collaborating tenants. For this purpose, we propose a new attribute function $UsedObject()$ which will be defined by the object owner and will be used by the collaborative tenant to define the global autorization.

4.3. AssignUser function

$AssignedUser_{(ta:TA;te:TE)}(u : U) \rightarrow \{True; False\}$, a boolean attribute function, mapping user to true or false, which means that the user attributes that represent the user who is responsible to achieve this task t will be defined by the tenant te in a confidential way. This compound attribute is used by the tenant responsible of the task to define the user attributes of the user who will perform this task in the local policy with a confidential and an autonomy way. For instance, the compound attribute $AssignedUser_{(T7;SH)}(u)$ will be defined by the tenant SH to specify the user attributes of the user responsible of the task $T7$. The indices used in this function are:

- $(ta : TA)$: the current task (the active task) of the workflow collaboration.
- $(te : TE)$: The tenant that will perform the this task ta (the task executor).
- The couple $(ta : TA, te : TE)$, means that the tenant te is responsible to accomplish the task ta.

4.4. UsedObject function

$UsedObject_{(ObjType;a:A;te:TE)}(o : O) \rightarrow \{True; False\}$, a boolean attribute function, mapping object to boolean $true$ or $false$, which means that the object attributes related to the object of the type ObjType will be defined by the tenant te in a confidential way. This compound attribute is used by the tenant provider of the resource to define the object attributes of this resource and the allowed action in the local policy with a confidential and an autonomy way. For instance, the compound attribute $UsedObject_{(MR;read;HH)}(u)$ will be defined by the tenant HH to specify the object attributes of the object (of the type MR) used in the collaboration. The indices used in this function are:

- $ObjType$: Set of objects that satisfy a common property are classified into an object type.
- $(a : A)$: the action related to the authorization.
- $(te : TE)$: the tenant that will share the object o in the collaboration.

Tasks	The Assigned Users	The Used Objects	Action
task(ti) = {interpret_scan} *workflow(ti) = {tenemo}* *previoustask(ti) = 'true'* *CSession(ti) = 'cs1'*	*MemberCS(u) = 'cs1'* *AssignedUser$_{(T5,SH)}$(u)='True'*	*SharedCS(o) = 'cs1* *UsedObject$_{(SCAN,Write,HH)}$(o)='True'*	*Write*
task(ti)={take_decision} *workflow(ti)={tenemo }* *previoustask(ti) = 'true'* *CSession(ti) = 'cs1'*	*MemberCS(u) = 'cs1'* ***AssignedUser$_{(T7,SH)}$(u)='True'***	*SharedCS(o) = 'cs1'* ***UsedObject$_{(MR,write,HH)}$(o)='True'***	*Write*
		SharedCS(o) = 'cs1' ***UsedObject$_{(SCAN,Read,HH)}$(o)='True'***	*Read*
		SharedCS(o) = 'cs1' ***UsedObject$_{(Video,Read,HH)}$(o)='True'***	*Read*

Figure 6. Example: Tasks definitions

4.5. A collaboration attributes based access control: C-ABAC model

Core C-ABAC is defined by the basic component sets, functions and authorization policy language given below:

- U, O, T and TE represent finite sets of existing users, objects, tasks and tenants respectively.
- A represents a finite set of actions available on objects. Typically $A = \{create; read; update; delete\}$.
- CTE represents finete set of collaborative tenants ($CTE \subseteq TE$).
- UA, OA and TA represent finite sets of user, object and task attribute functions respectively.
- For each $att \in UA \cup OA \cup TA$, $range(att)$ represents the attribute's range, which is a finite set of atomic values.
- $attType : UA \cup OA \cup TA \rightarrow \{set; atomic\}$, specifies attributes as set or atomic values.
- $Collabors : (cte : CTE) \rightarrow TE$, specifies the tenants that will use this collaborative tenant cte.
- Each attribute function maps elements in U to an atomic value or a set
  - $\forall ua \in UA.\quad ua : U \rightarrow Range(ua)$ if $attType(ua) = atomic$
  - $\forall ua \in UA.\quad ua : U \rightarrow 2^{Range(ua)}$ if $attType(ua) = set$
- Each attribute function maps elements in O to an atomic value or a set
  - $\forall oa \in OA.\quad oa : O \rightarrow Range(oa)$ if $attType(oa) = atomic$
  - $\forall oa \in OA.\quad oa : O \rightarrow 2^{Range(oa)}$ if $attType(oa) = set$
- Each attribute function maps elements in T to an atomic value or a set
  - $\forall ta \in TA.\quad ta : T \rightarrow Range(ta)$ if $attType(ta) = atomic$
  - $\forall ta \in TA.\ ta : T \rightarrow 2^{Range(ta)}$ if $attType(ta) = set$
- $UOwner : (u : U) \rightarrow TE$, required attribute function mapping user u to owner tenant te.
- $OOwner : (o : O) \rightarrow TE$, required attribute function mapping object o to owner tenant te.
- $TOwner : (t : T) \rightarrow TE$, required attribute function mapping task t to owner tenant te.
- $UAOwner : (uatt : UA) \rightarrow TE$, meta attribute function, mapping user attribute ua to attribute owner tenant te.
- $OAOwner : (oa : OA) \rightarrow TE$, meta attribute function, mapping object attribute oa to attribute owner tenant te.
- $TAOwner : (ta : TA) \rightarrow TE$, meta attribute function, mapping task attribute ta to attribute owner tenant te.

- $ua(u:U)$ is defined only if $(UAOwner(ua) = UOwner(u)) \cup (UOwner(u) \in Collaborators(UAOwner(ua)))$.
- $oa(o:O)$ is defined only if $(OAOwner(oa) = OOwner(o)) \cup (OOwner(u) \in Collaborators(OAOwner(oa)))$.
- $ta(t:T)$ is defined only if $(TAOwner(ta) = TOwner(o))$.
- $AssignedUser_{(ta:TA;te:TE)}(u:U) \rightarrow \{True; False\}$, a boolean attribute function, mapping user u to $true$ or $false$, which means that the user attributes that represent the user who is responsible to achieve this task ta will be defined by the tenant te in a confidential way.
- $UsedObject_{(ObjType;a:A;te:TE)}(o:O) \rightarrow \{True; False\}$, a boolean attribute function, mapping object o to $true$ or $false$, which means that the object attributes related to the object of the type $ObjType$ will be defined by the tenant te in a confidential way.
- An authorization that decides on whether a user u can access an object o in a particular task t for the action a, is a boolean function of u, o, and t attributes: Rule: $authorization_a(u; o; t) \rightarrow f(UA(u); OA(o); TA(t))$, with the additional required condition that $(UOwner(u) = OOwner(o) = TOwner(t)) \cup (UOwner(u) \in Collaborators(TOwner(t)) \cap OOwner(o) \in Collaborators(TOwner(t)))$.

4.6. Example: C-ABAC Authorizations

Let us consider a telemedicine scenario where the School Hospital (SH), the Emergency Medical Services (EMS), and the Home Hospital (HH) are three collaborating organizations. In this example, we apply the C-ABAC model on the telemedicine use case a telemedicine previously depicted by specifying authorizations for the tasks $interpret\_scan$ and $take\_a\_decision$ as shown in figure 7. First, for each task we define a set of task attributes, Set of user attributes that represent the user who is supposed to achieve this task and set of access permissions related to this task. a permission is an action on object. an object is defined by a set of object attributes. For instance $(write, MR)$, $(read, scan)$ and $(read, video)$ are three permissions related to the task $take\_a\_decision$.

Access control authorizations in C-ABAC model are defined by the following the formalism shown in the figure 7:

The authorization $Authorization_{write}(ti; u; o)$ that is shown in the figure 7 matches to 'the rule the radiologist interprets the scan images' as specified at the first line in figure 6. This authorization is defined in the collaborative tenant $CT1$ and is composed of a set of task attributes, user attributes and objects attributes. This authorization is valid for the action $write$ if only if : (1) The instance ti is instantiated of the task $interpret\_scan$; (2) the task instance ti belongs to the workflow $tenemo$; (3) the previous task instances of the ti are accomplished; (4) There is a collaborative session in which the task runs; (5) The user u is member of the collaborative session cs; (6) The object o is shared in the session cs; (7) The tenant SH authorizes his user u to perform the task $T5$; (8) The tenant HH shares the object o of the type $ObjectType$ with others collaborating tenants for the action $write$.

The attributes $AssignedUser$ and $UsedObject$ are defined and evaluated in the tenant SH and HH respectively. This attribue $AssignedUser_{(T5;SH)}(u) = True$ if only if: (1) the user u plays the role radiologist; (2) u is at least level 1 of the neurology expertise; (3) u is at least level 2 of the radiology expertise; (4) u is at least level 0 of the cardiology expertise. The attribute $UsedObject_{(SCAN;Write;HH)}(o) = True$ if only if: (1) the object o of the type $Scan$; the sensitivity class of object o is less than or equal to $class2$.

5. Implementation

5.1. System architecture

OpenStack is a robust open-source IaaS software for building public, private, community or hybrid clouds. OpenSteck is adopted by many cloud providers such as Rackspace, IBM and RedHat. OpenStack contains the following components: Nova, Swift, Glance, Cinder, Keystone, and Horizon. Each component acts as a service which communicates with other services via message queues. Keystone provides authentication and authorization for all OpenStack services. In our work, we focus on the Swift object storage. Swift is a multi-tenant, highly scalable and durable software defined storage system designed to store files, videos, virtual machine snapshots and other unstructured data [7]. It allows building, operating, monitoring, and managing distributed object storage systems that can scale up to millions of users.

The Account Server is responsible for listings of containers, while Container Server is responsible for listings of objects. A container is a mechanism that stores data objects. An account might have many containers, whereas a container name is unique. A user represents the entity that can perform actions on the object in the account. Each user has its own account and is associated to a single tenant. Swift uses the access control lists (ACL) to manage the access permissions. In fact, the ACL model defines static access rules. It is not suitable for collaborative environment. In this paper, we implemented the C-ABAC model on the swift storage

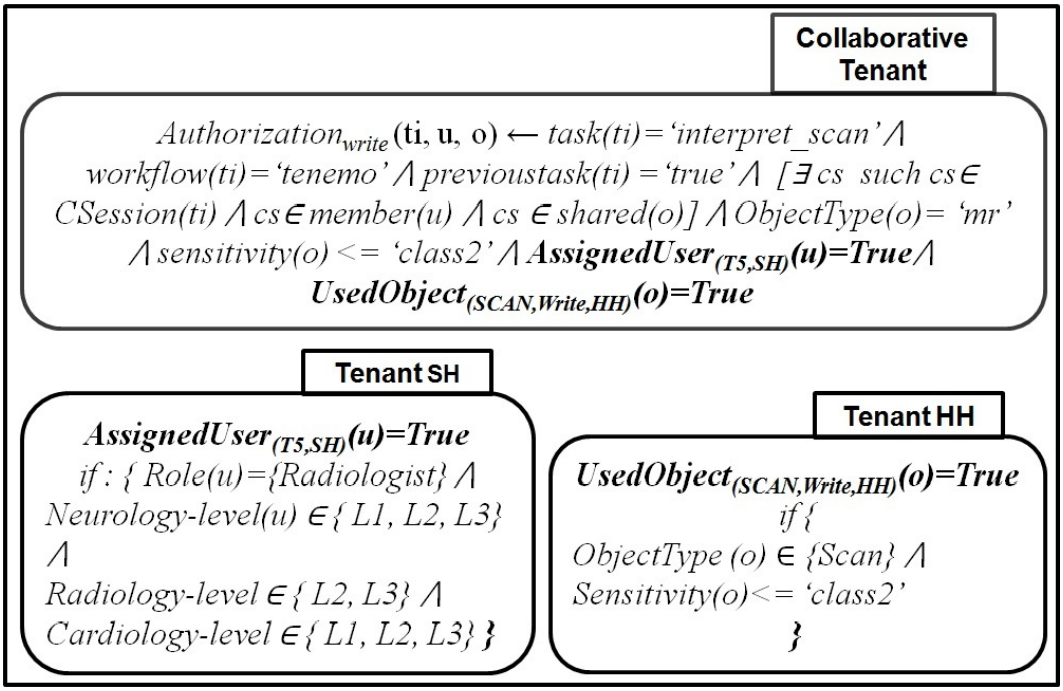

Figure 7. Example: C-ABAC Authorizations

component. This component acts as a service that communicates with other components (Nova volume, nova compute, nova network, glance and keystone) via message queues. These components are loosely coupled. Keystone is the identity service used by OpenStack for authentication and authorization. It provides a token signed by each user's private key.

Let us consider the telemedicine scenario where the School Hospital (SH), the Emergency Medical Services (EMS), and the Home Hospital (HH) are three collaborating organizations. These organizations share a common private cloud openstack. We consider that these organizations use the swift component for the storage service. In this use case, each organization is assigned to a swift account. (e.g. the accounts ACC_SH, ACC_EMS and ACC_HH represent the organizations SH, EMS and HH respectively).

This cloud provides a service of collaborative sessions for these organisations. This service allows a group of users, from different tenants, to collaborate in order to observe and treat a patient admitted in the Home Hospital (HH) emergency. In this example, we have a collaborative session $CS1$ of a telemedicine type. During a collaborative session, users may intervene dynamically without a prior knowledge of which user will access which object.

In order to support C-ABAC Model in the OpenStack Swift environment and overcome the limitations of Swift ACL, we propose to extend the Swift component by implementing a new C-ABAC Module (Figure 8). The C-ABAC module is composed of five components:

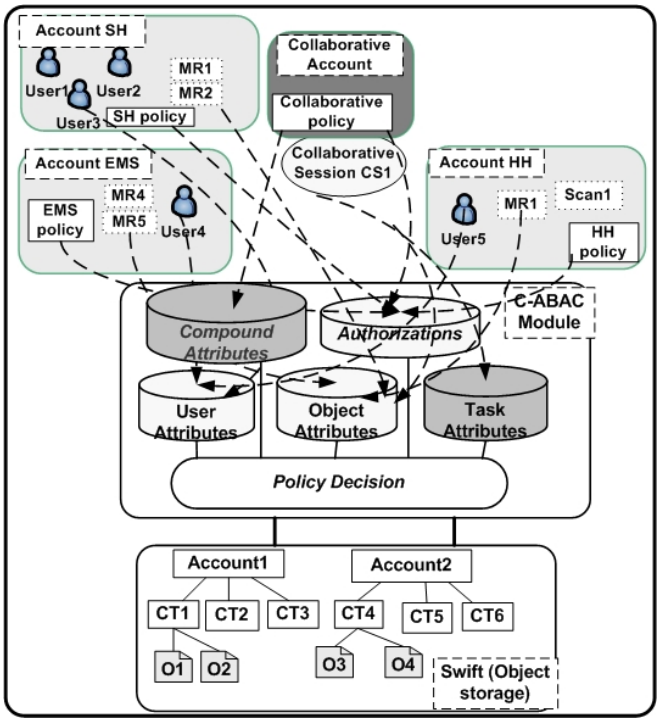

Figure 8. The system architecture

User attributes, object attributes, task attributes, authorizations and the policy decision component. In the following, we describe each of these components:

- **User attributes:** The security administrator defines the user attributes as a function that takes user as input and returns a value from the attribute's range. $(user1 : attr1 : val1)$ means that for the user $user1$ the value of the attribute $attr1$ is $val1$. For example, a user attribute function such as $Role \in UATT$ maps $user1 \in U$ to a value $neurologist$. Furthermore, the cloud administrator defines the attribute function $UOwner$ to specify the user owner. For instance $(user1 : UOwner : ACC\_SH)$ means that the user $user1$ is owned by the account ACC_SH. Finally, the administrator defines the attribute function $JoinCS$ to specify which users could join the collaborative sessions. The value of this attribute is either $true$ or $false$. $(user1 : JoinCS : true)$ means that the user $user1$ could participate in the collaboration.

- **Object attributes:** The tenant administrator assigns the object attributes as a function that takes object as input and returns a value from the attribute's range. $(obj1 : attr1 : val1)$ means that for the object $obj1$ the value of the attribute $attr1$ is $val1$. Furthermore, the cloud administrator defines the attribute function $OOwner$ to specify the object owner. For instance $(MR1 : UOwner : ACC\_HH)$ means that the object $MR1$ is owned by the account ACC_HH. Finally, the administrator defines the attribute function $SharedCS$ to specify which objects could be shared in the collaborative session. For example, $(Per\_info1 : SharedCS : false)$ means that the object Per_info1 (personal information) could not be shared in the collaborative session.

- **Task attributes :** The administrator assigns the task attributes as a function that takes task instance as input and returns a value from the attribute's range. $(ti1 : attr1 : val1)$ means that for the task instance $ti1$ the value of the attribute $attr1$ is $val1$. Furthermore, the cloud administrator defines the attribute function $TOwner$ to specify the task owner. For instance $(ti1 : TOwner : ACC\_EMS)$ means that the task instance ti is performed by the account ACC_EMS.

- **Compound attributes:** The administrator of the local tenant defines the new attributes $AssignedUser$ and $UsedObject$ with a confidential and an autonomy way. These compound attributes are specified here as follows: $UsedObject|SCAN|Write|HH : -o : ObjectType : SCAN \wedge o : sensitivity : class0|class1|class2$, which means that this attribute is true if only if: (1) the object o of the type $Scan$; the sensitivity class of object o is less than or equal to $class2$.

- **Authorizations:** The administrator specifies the authorizations policy. In our scenario, we consider that each tenant defines its policy rules. Note that at this level, we suppose that the security policy rules are valid and conflict-free. The policy rules are specified here as follows: $write - ti : task : interpret\_scan \wedge ti : workflow : tenemo \wedge ti : previoustask : true \wedge ti : CSession : cs1 \wedge u : memberCS : cs1 \wedge o : sharedCS : cs1 \wedge u : AssignedUser|T5|SH : True \wedge o : UsedObject|SCAN|Write|HH : True$, which means that for the action $write$', this authorization is valid if only if : (1) The instance ti is instancied of the task $interpret\_scan$; (2) the task instance ti belongs to the workflow $tenemo$; (3) the previous task instances of the ti are accomplished; (4) There is a collaborative session in which the task runs; (5) The user u is member of the collaborative session cs; (6) The object o is shared in the session cs; (7) The tenant SH authorizes his user u to perform the task $T5$; (8) The tenant HH shares the object o of the type $SCAN$ with others collaborative tenants for the action $Write$.

- **Policy decision:** This component is responsible for evaluating the access request to the resources in the collaborative session based on the collected attributes values and authorizations. When a user sends a request to access a resource stored in the cloud swift, the policy decision component evaluates this request according to the policy rules in order to decide whether the user is authorized to access this resource or not.

5.2. Enforcement model

A general authorization process for Swift component with C-ABAC module is illustrated in figure 9. When the user $user1$ attempts to access the resource $MR1$ stored in the swift. First, (1) The user requests keystone to get his/her token. (2) Keystone generates a token and sends it to the user. (3) The user sends a request to ABAC module by using his/her token to access the resource $MR1$. The Policy decision component receives this request to evaluate it.
(4) During the evaluation process, the policy decision component requests the components: user attributes, object attributes and task attributes (5) to receive user1's attributes, $MR1$'s attributes and the attributes related to the collaborative session wherein this user is member.
(6) The policy decision component requests the authorizations component and (7) receives all the policy rules stored in this component. These attributes and

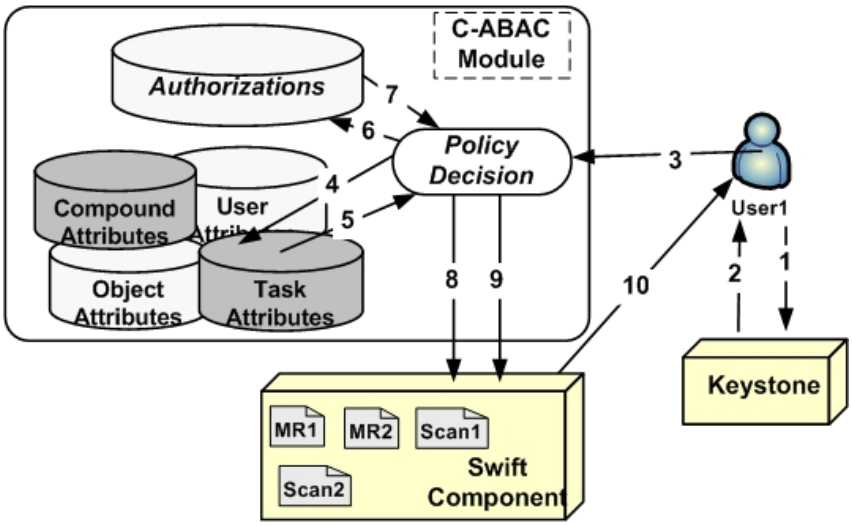

Figure 9. The enforcement model

policy rules will be used by the policy decision to evaluate access request in order to decide whether the user is authorized to access this resource or not. (8) the policy decision will execute an ACL command to assign the authorization decision (permit or deny) to the user in the swift environment. (9) The policy decision component executes a swift API command in the swift component using *user*1's token in order to send the *user*1's access request to swift. (10) *user*1 access to the resource *MR*1 if the authorization decision is permitted.

5.3. Evaluation

In this paper, we implement the ABAC and C-ABAC on the swift storage component of openstack. Our experiments were run on a virtual machine with the following characteristics (Memory 1024MB, 2 cores CPU, Hard Disk 30GB). We consider the download time of a Swift object using ABAC model and C-ABAC Model. We observe that the performance of enforcing our approach depends on many factors, such as numbers of rules, number of attributes and number of concurrent collaborative sessions. In our analysis, we have used a synthetic dataset that contains up to 2000 rules, 2500 attributes and 25 concurrent collaborative sessions.

Figure 10(a) shows that the average time to authorize the access to a Swift object with ABAC model increases with 13.4% and 22.7% for policies of 400 and 2500 rules respectively using the C-ABAC model. The waiting time for getting a policy decision becomes larger when there are too many authorizations to be collected. We acknowledge that our implementation works well for a large number of authorizations.

Furthermore, we compute the running time for access/deny decisions to a Swift object using ABAC and C-ABAC model for 400 rules and for 500 to 2500 attributes. Figure 10(b) shows that the average time for download of a Swift resource with ABAC model increases with 7.6% and 19.6% for 500 and 2500 user attributes assignments using C-ABAC Module. We acknowledge that our implementation works well for a large number of authorizations.

Finally, we compute the running time for access/deny decisions to a Swift object using ABAC and C-ABAC model for 400 rules, 500 Attributes and number of concurrent collaborative sessions with 10 to 50 active ones.

Figure 10(c) shows that the average time for access/deny decisions to Swift resources using ABAC model increases with 20.1% and 34.7% for 10 and 50 concurrent collaborative sessions respectively using the ABAC Module. We observe that our implementation works well for a medium number of active concurrent collaborative sessions. The overhead reaches 34.7% in an unusual situations where there are 50 concurrent parallel collaborative sessions.

6. CONCLUSION

In this paper, we present a novel ABAC based access control model called (C-ABAC). C-ABAC enables to ensure access control in collaboration between tenants in the cloud. C-ABAC allows the collaborating tenants to specify a global policy in a specific central point. It supports multiple concepts such as: task and workflow. The suggested model ensures the autonomy of tenant and preserves the confidentiality of each tenant object. Finally, an architecture that integrates C-ABAC in the storage level of the cloud platform OpenStack has been described. The implementation results have shown that the suggested approach has a very limited overhead.

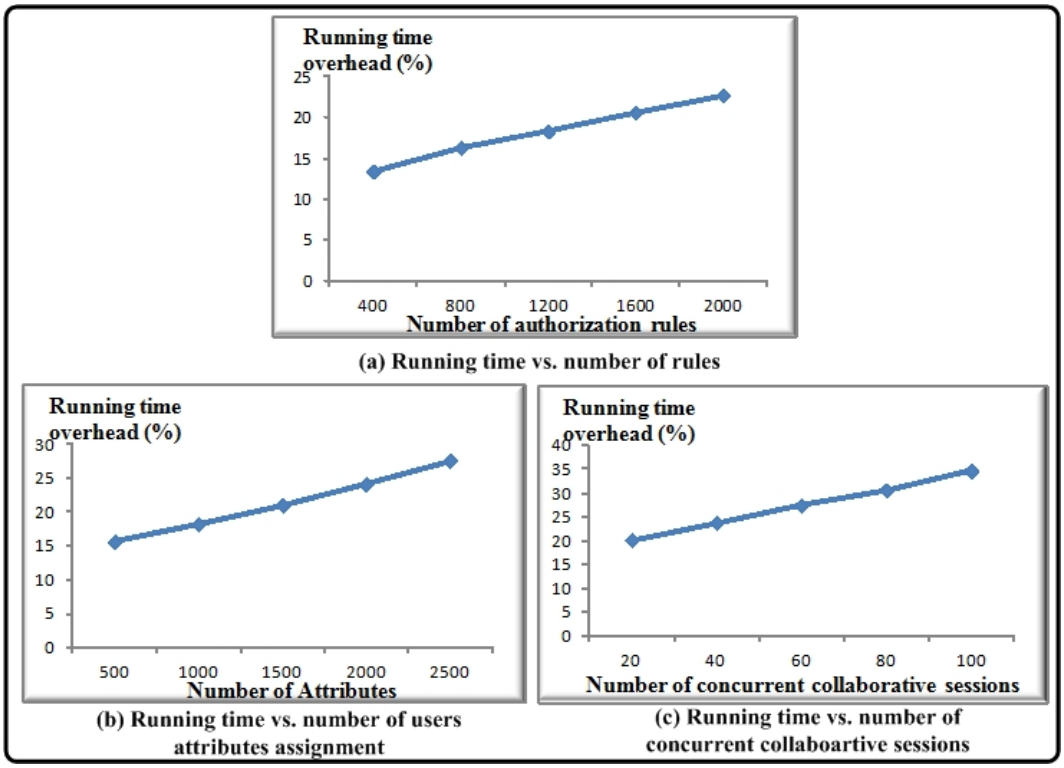

Figure 10. Running time overhead for access/deny decisions

References

[1] P. Mell and T. Grance. The NIST Definition of Cloud Computing. NIST Special Publication 800-145 (Draft). Retrieved September 10, 2011, from http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145-cloud-definition.pdf.

[2] J. M. A. Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray. Toward a multi-tenancy authorization system for cloud services. IEEE Security and Privacy, vol. 8, no. 6, pp. 48-55. 2010.

[3] B. Tang, and R. Sandhu. A Multi-Tenant RBAC Model for Collaborative Cloud Services. in PST, pp. 229-238, 2013.

[4] H. Takabi, J. B. D. Joshi, and G. J. Ahn, .SecureCloud: Towards a Comprehensive Security Framework for Cloud Computing Environments. In Proc. of the 1st IEEE International Workshop Emerging Applications for Cloud Computing, pp. 393-398, Seoul, South Korea, 2010.

[5] A. Tanvir, A. R. Tripathi. Specification and verification of security requirements in a programming model for decentralized CSCW systems. ACM Trans. Inf. Syst. Secur. 10(2) (2007).

[6] OpenStack cloud platform. http://www.openstack.org/. Accessed: 2016- 10- 05.

[7] OpenStack Swift Architecture. https://swiftstack.com/openstack-swift/architecture/. Accessed: 2016- 10- 05.

[8] Y. Zhang, R. Krishnan, R. Sandhu. Secure information and resource Sharing in cloud. CODASPY, pp. 131-133, 2015.

[9] Jin, X., Krishnan, R., Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. DBSec 12, pp. 41-55 (2012).

[10] E. Yuan, and J. Tong. Attributed Based Access Control (ABAC) for Web Services. ICWS IEEE Computer Society, pp. 561-569. 2005.

[11] N. Pustchi, R. Sandhu. MT-ABAC: A Multi-Tenant Attribute-Based Access Control Model with Tenant Trust. NSS. pp. 206-220. 2015.

[12] R. Thomas. TMAC: A primitive for Applying RBAC in collaborative environment. 2nd ACM, Workshop on RBAC, pp. 13-19, Fairfax, Virginia, USA, November 1997.

[13] R. Thomas and R. Sandhu. Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management. 11th IFIP WorkingConference on Database Security, Lake Tahoe, California, USA, 1997.

[14] O.H. Sejong, S.Park. Task-role-based Access Control Model. In: Information Systems, 28(6): pp. 533-562, 2003.

[15] X. Jin, R. Krishnan, R. Sandhu. Role and attribute based collaborative administration of intra-tenant cloud IaaS. CollaborateCom. pp. 261-274. 2014.

[16] P. Biswas, F. Patwa, R. Sandhu. Content Level Access Control for OpenStack Swift Storage. CODASPY. 123-126. 2015.

[17] P. Biswas, R. Sandhu, R. Krishnan. An Attribute Based Protection Model for JSON Documents. In *NSS. 303-317*, 2016.

[18] D. Lin, P. Rao, E. Bertino, N. Li, J. Lobo, Policy decomposition for collaborative access control, SACMAT 2008: 103-112.

[19] A. Madani, M. Erradi, Y. Benkaouz. Access Control in a Collaborative Session in Multi Tenant Environment. 11th International Conference on Information Assurance and

Security, Marrakech, December 2015.

[20] B. Tang, R. Sandhu, Q. Li. Multi-tenancy authorization models for collaborative cloud services. in IEEE International Conference on Collaboration Technologies and Systems, 2013.

[21] M. A. Madani, M. Erradi, Y. Benkaouz. A Collaborative Task Role Based Access Control Model. Journal of Information Assurance and Security, vol. 11, no. 6, pp. 348-358, 2016.