Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 11 (2016) pp. 348-358

A Collaborative Task Role Based Access Control Model

Mohamed Amine Madani, Mohammed Erradi, Yahya Benkaouz

Networking and Distributed Systems Research Group, SIME Lab,
ENSIAS, Mohammed V University in Rabat, Morocco
amine.madani@um5s.net.ma, mohamed.erradi@gmail.com, y.benkaouz@um5s.net.ma

Abstract: **Cloud computing allows to move computing and storage components from individual systems into the cloud, which provides software and hardware services over the Internet. A collaborative application is among software services that can be provided by the cloud computing to enable collaboration among users from the same or different tenants. During such 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 sessions. This paper proposes a Collaborative Task Role-Based Access Control CTRBAC[1] model to ensures 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: Collaboration in multi-tenant environment, active and passive access control and collaborative sessions. The suggested model has been implemented using Swift component in the open source cloud-computing platform "OpenStack".**

Keywords: access control; cloud multi-tenant; collaborative task Role-Based Access Control; OpenStack.

I. Introduction

Cloud computing allows to move computing and storage components from individual systems into the cloud, which provides software and hardware resources over the Internet. The US National Institute of Standards and Technology (NIST) [1] defines cloud as follows:"*Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and re-leased with minimal management effort or service provider interaction*".
The cloud model has several characteristics: on-demand self-service, ubiquitous network access, resource pooling, rapid elasticity and measured service. In cloud environment, we distinguish between two deployment models: a single tenant model and a multi-tenants model. In the first one, each customer gets a separate instance of the software that runs on a logically isolated hardware environment. The second one is defined as the capability of a single instance of a software application to serve, simultaneously, multiple clients (tenants). It allows the separation of tenants in order to secure the access to customers resources in the cloud.
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. 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. This collaborative session is an abstract entity, comprising a set of users, called members of the session, playing the same or different roles. These users may have concurrent access to shared objects in this session depending on their roles.
Using collaborative sessions in cloud environment requires a fine-grained access control model. This model should be flexible enough in order to support the requirements of the collaborative session. In this model, each tenant may share several resources with other tenants belonging either to the same issuer or to a different issuer in order to perform a common task. In this paper, we propose an approach that ensures access control to the shared resources in a collaborative session in multi-tenants environments. This means that collaborating users and the shared resources belong to the same or different tenants. We suggest CTRBAC, the Collaborative Task Role-based Access Control, as an extended version of the CRBAC model [21]. In CTRBAC, new entities were added in order to support together: collaboration in multi-tenant environment, active and passive access control and collaborative sessions. Moreover, we propose an administrative model for this approach. The suggested model has been implemented using Swift component in the open source cloud-computing platform OpenStack.
The rest of this paper is organized as follows: Section 2 presents a use case of collaborative session in cloud environment. Section 3 discusses the related work in the context of access control models in cloud environments. Section 4 formally presents the proposed model CTRBAC. Section 5 presents the administrative model. Section 6 describes CTRBAC implementation. Finally, we conclude in Section 7.

[1]This work is supported by the BMBF (PMARS Programme) and the DAAD (German-Arab Transformation Partnership)

II. CASE STUDY: A COLLABORATIVE APPLICATION FOR TELEMEDECINE

In this section, we show a case study which consists in a telemedicine use case 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 three SHs 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 example, 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.*

In this example, we assume that each issuer has its own administrator which could define the security policies belonging to this issuer independently from the other collaborating issuers. This administrator could perform many administrative operations, such as:

- Add a new tenant;
- Delete a tenant;
- Add/delete users to/from the tenant;
- Create roles, collaborative session templates, workflows and tasks;
- Assign users to the roles
- Assign tasks to roles
- Assign the permissions to tasks
- Assign the objects to object types

The example of the workflow schema (figure 2) shows a diagnosis in neurology field and its emergencies. This workflow is defined as a set of tasks that are connected in order to take the decision on the type of care to apply to the patient. In this use case, we consider that each task in the collaborative session will be active only if the previous tasks are performed.
The task assignment table (Table 1) describes which tasks are assigned to which roles in the collaborative session. Moreover, the issuer administrator could assign a set of permissions to each task. For instance, the permissions *(read, scan_image)* and *(write, scan_imag)* are assigned to the task

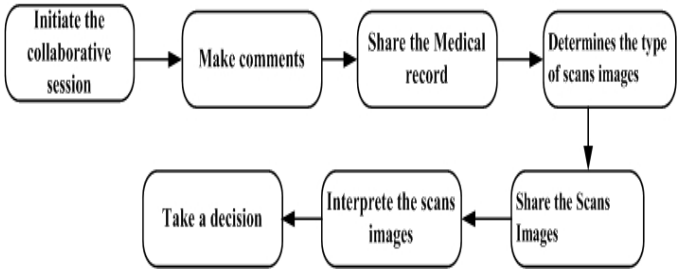

Figure. 2: A workflow schema for diagnosis in neurology

Role	*Assigned task*
Doctor_EMS	*ta1: Initiate the collaborative session*
Doctor_HH	*ta2: Make comments*
Doctor_HH	*ta3: Share the Medical record*
Neurologist	*ta4: Determines the type of scans images*
Doctor_HH	*Ta5: Share the Scans Images*
Radiologist	*Ta6: Interpret the scans images*
Neurologist	*Ta7: Take the decision*

Table 1: Task assignment table

Interpret the scans images. The considered scenario involves different cross-tenant access rules. Among these rules, we find the following:

- The *doctor_EMS* in the tenant *Emr* has the authorization to create a collaborative session and invite users to join this session.
- *User5* has the right to share the patient's medical record in the collaborative session.
- All members of the session must access to the shared medical record to perform the medical diagnosis.
- Non-members of the collaborative session *CS1* do not have the rights to participate in this session.
- Only the neurologist in this collaborative session has the permission to decide on the type of care to apply to the patient.
- The radiologist in the collaborative session has the right to interpret scans images shared in this session.
- The permissions associated to each task will be active only if the previous tasks are performed.

III. RELATED WORK

In the Role Based Access Control Model (RBAC) [1], [2], [3], the permissions are assigned with roles, and users are assigned to appropriate roles. This model significantly reduces administration overheads. In OrBAC (Organization Based Access Control) [10] model, the expression of an authorization policy is centered on the concept of organization. An organization is a structured group of active entities, in which subjects play specific roles. An activity is either one or more actions, a view can be either one or more objects, and a context is designed to handle dynamic parameters of a policy. However, these models do not separate task from role and do not support workflow.

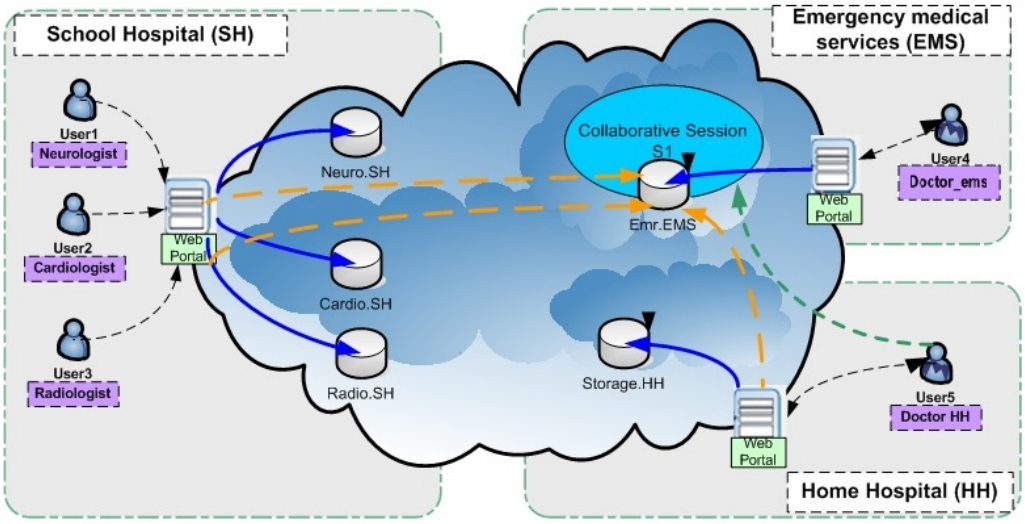

Figure. 1: A collaborative session in cloud environment

In the Task based access control [4] (TBAC), the permissions are granted in steps that are related to the progress of tasks. The TRBAC [18] model is constructed by adding task 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 dynamic collaboration by using the collaborative sessions. Moreover, the notion of team used in TMAC is not dynamic.

On the other hand, Cuppens et al proposes in [11] an approach to secure dynamic session. This approach defines a concept called "switchability "where means that a user can share his session with another user from the same or different organization. However, the dynamic session proposed in this approach is different from the concept of collaborative session proposed in our approach: All members of the dynamic session play the same roles in this session. While in the collaborative session, the users members may play the same or different roles. These users can perform specific tasks, by accessing shared resources in order to achieve a common goal.

Other access control approaches have been suggested to secure resources in cloud Environment [7], [8], [9], [13], [14]. Calero and al [7] suggests a multi-tenancy authorization system. His work is based on role-base access control with a coarse-grained trust relation, path-based object hierarchies and hierarchical RBAC. Calero et al [7] assumes that each issuer may use several cloud services and could collaborate with other issuers.

Tang et al [8] proposes in a multi-tenancy authorization system (MTAS) model 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 accesses to a trustees resources.

The Multi-Tenant Role-Based Access Control (MT-RBAC) proposed by Tang et al in [9] is a model to provide 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, and this trustee assigns their users to these trusters roles so these users can access to the truster's resources by activating the trusters roles.

Other access control approaches [14], [15] have been proposed to ensure access control in collaborative multi-clouds environments. Authors present in [14] a distributed access control architecture for multi cloud environments in order to support dynamic collaboration between clouds. This approach requires pre-establishing collaboration agreements (SLA) among clouds providers. Another approach [15], that doesn't require prior agreements between the cloud service providers, makes use of a proxy in order to support collaboration among heterogeneous cloud.

After analyzing these access control approaches related to the collaboration in the cloud environment, we noticed that these approaches do not support the resources sharing rules. These rules are used to secure resources sharing among users in collaborative sessions. These rules become more complex if we consider the members of the collaborative session are from different tenants and the shared resources are owned by different tenants. For instance: The user *U1* playing the role *R1* and member of the collaborative session *CS1* is authorized to access the object *O1* that is shared in this collaborative session *CS1*. And if we have a user *U2* that plays the same role *R1* and he is not member of the session *CS1*, then this user will not be authorized to access to this object O1. Also if we have an object O2 that is not shared in this session *CS1*, then the user *U1* will not be authorized to access to this object *O2*.

In this paper, we formally define a new model that ensures access control to shared resources in a collaborative session in multi-tenants environments. This model supports:

- Collaboration in multi-tenant environment: Collaborations among tenants require an adaptive and a flexible access control model. In this model, each tenant may share several resources with other tenants belonging either to the same issuer or to a different issuer in order to

perform a common task.

- Collaborative sessions: We need to have a fine-grained access control policies, in order to support the requirements and the management of the collaborative session like: Only users members in the collaborative session could: access to the shared resources in such session, join/leave the collaborative session and share/unshare resources in/from the session.

- Task and workflow: This model should enable the granting and revoking of permissions to be automated during the progression of the running tasks. Also, it should support active and passive access control.

IV. CTRBAC: COLLABORATIVE TASK ROLE BASED ACCESS CONTROL

In our previous works, we have proposed CRBAC [21] Collaboration-Role Based Access Control model as an extended version of RBAC Model, in which new entities were added in order to support collaborative sessions in multi-tenant environments. This model introduces the entities: *Collaborative sessions, collaborative session templates, tenants* and *issuers*. Also, a new trust relation among tenants was introduced. In this work we present CTRBAC Collaboration-Task-Role Based Access Control model: an extension of CRBAC Collaboration-Role Based Access Control model model, in order to consider tasks and their workflow during a collaborative session in multi-tenants environments. The suggested CTRBAC introduces the entities *(tasks, tasks instances, workflow schema, workflow instances)* to the original CRBAC model to allow a dynamic access control during a collaborative session. Moreover, this model CTRBAC supports active and passive access control and enables the granting and revoking of permissions to be automated with the progression of the tasks in workflow systems.
In this model, as shown in Figure 3, we have the following entities: we have many entities: *Issuers (I), tenants (T), collaborative sessions (CS), collaborative session templates (CST), roles (R), tasks (TA), task instances (TI), workflows schema (WS), workflows instances (WI), permissions (P), users (U), sessions (S), actions (C), objects (O)* and *objects types (B)*. An issuer has one or several tenants. A tenant is associated to one issuer by a many-to-one relationship. Further, there are many-to-one relationships from different entities to their owner tenants. These entities are described in the following:
ISSUERS. An issuer is an organization or an individual that uses the cloud services. It is a client who is able to administer its own tenants in the cloud. For instance, in the previously described scenario, the SH is an issuer in a single private cloud service.
TENANTS. A tenant is a virtual partition of a cloud service provided by the cloud provider to the issuer. An issuer is associated to multiple tenants while a tenant belongs to a single issuer. For instance, tenants neuro, cardio and radio are examples of tenants that are belonging to the issuer school hospital.
COLLABORATIVE SESSION TEMPLATES. It defines a pattern for a collaboration activity[15]. A collaborative session is created and started by instantiating its template. We can have different kinds of collaborative session templates: (public session template, private session template, emergency in the neurology, emergency in the cardio). A collaborative session template belongs to a single tenant while a tenant may have multiple templates.
A Collaborative Session Template is defined as the quadruplet *(cst, R, B, TA)*:

- *cst: Template ID;*
- *R: Set of Roles;*
- *B: Set of Object Types;*
- *TA: Set of tasks performed in the session.*

COLLABORATIVE SESSIONS. A collaborative session is the basic entity in our model. It is an abstract entity, comprising a set of users (called members of the session), that performs specific task, by accessing to shared resources to achieve a common goal. Each collaborative session is an instance of the collaborative session template. This kind of session is characterized by a set of parameters: id, members participating to the session, shared objects and a set of tasks. A collaborative session is defined as the quadruplet *(cs, U, O, TI)*:

- *cs: The collaborative session ID which is an instance of the template cst;*
- *U: Set of users (u is an instance of the role r);*
- *O: Set of objects shared in this collaborative session (o is an Instance of the object type b);*
- *TI: Set of task instances performed in the session.*

As shown in figure 4, we have a collaborative session template cst from which both the Collaborative sessions cs1 and cs2 are instantiated. In this template cst, we define a set of roles R1, R2 and R3. For each role, we assign a set of permissions. A permission is defined in the template as an action on an object type.
When a user instantiate a collaborative session cs1 from the template cst, the entities users and objects in the session cs1 will be instantiated from the entities roles and objects type respectively in the template. For example, the users u2 and u3 in the collaborative session cs1 are instantiated from the role R2, which means that these users play the role R2 and are members of this session cs1. The same for the object obj1, which is instantiated from the object type ObjType1.
ROLES. A role is a collection of users that have the same job function in the tenant. A tenant may own multiple roles while a role is associated to a single tenant.
TASKS. Task [6] is a fundamental unit of a business work or a business activity. Job function is another expression of task. Share the Scans Images, Interpret the scans images and Take the decision are examples of tasks. Tasks are assigned to users by their job positions or business roles. Tasks can be divided into two class:

- The first class is the task that does not belong to workflow ($ta \notin WFS$).

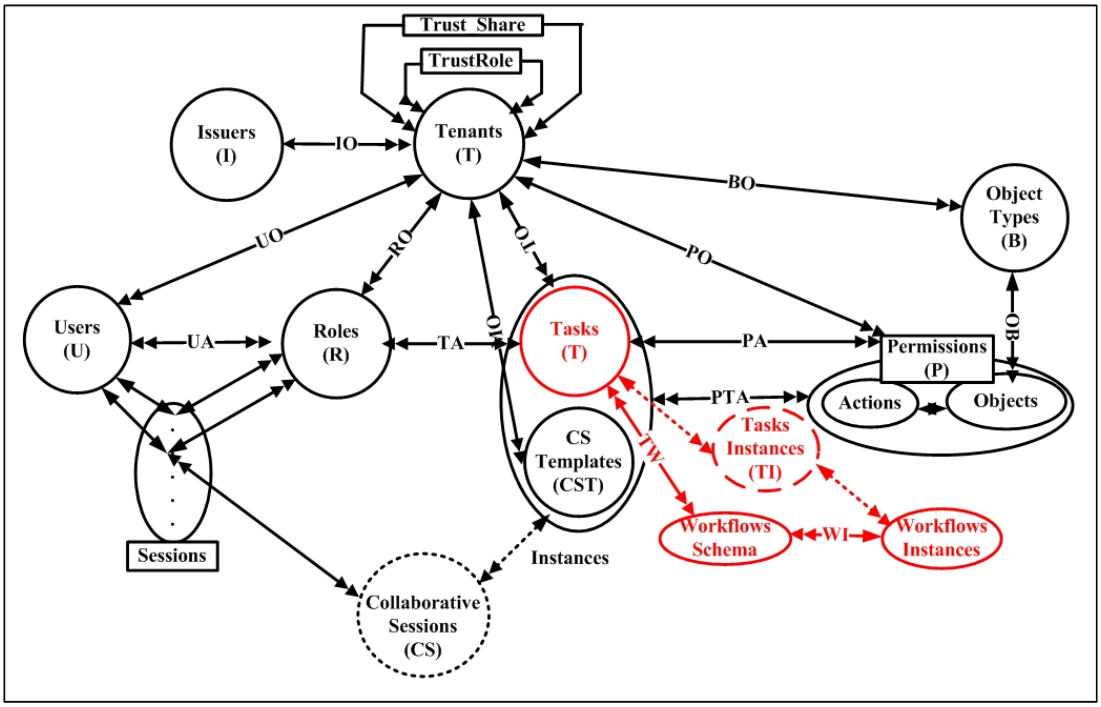

Figure. 3: Collaborative Task Role Based Access Control model

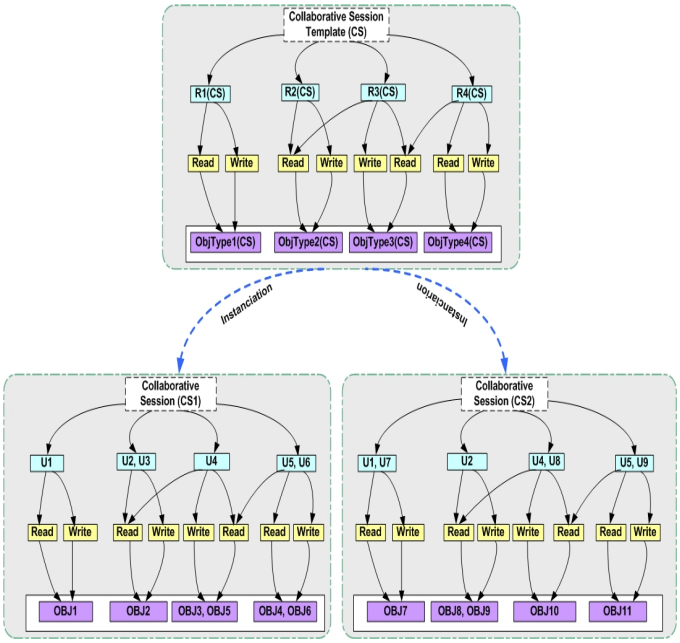

Figure. 4: Collaborative session template

- The second one is the task that belongs to workflow ($ta \in WFS$).

WORKFLOWS. The workflow is defined as a set of tasks that are connected to achieve a common goal. In general, it means a product or a methodology for supporting a business process in the enterprise environment. For instance, diagnosis in neurology is an example of workflows schema defined in the telemedicine use case.

USERS. A user is the entity that can perform actions on the object in the tenant. A user might have different users profiles in different tenants. The relationship between the user and the collaborative session: member(u, cs), which means that the user u is member of the collaborative session cs.

SESSIONS. An individual session (as defined in RBAC model [2]) is an instance of activity established by a user. Each individual session is a mapping between a user and an activated subset of roles that are assigned to the user. A collaborative session is associated to multiple individual sessions while an individual session could be connected to one or multiple collaborative session.

OBJECTS. An object is the resource that the security policy attempts to control its access from unauthorized users. An object could be data objects, files, directories, devices, and ports. The relationship between the object and the collaborative session: shared(o, cs), means that the object o is shared in the collaborative session cs.

OBJECT TYPES. CTRBAC categorizes objects into views. An object type represents a kind of resources such as s-can_image.

In this paper, we consider a multi-tenant environment, where collaborating users and the shared resources might belong to the same or different tenants. As shown in figures 4, the user5 from the home hospital tenant attempts to join a collaborative session that turns in the EMS tenant. This user could share some HHs resources in this collaborative session. In order to support collaboration and shareability of resources among tenants, we reuse the role trust relation as defined in [5] and we define a new trust relation *TrustShare()*.

A. *Trust role relation*

As defined in MT-RBAC [5] the tenant trust relation $(TT) \subseteq T \times T$ is a many-to-many reflexive relation between the truster Tr and the trustee $Te : TrustRole(tr, te : T) \rightarrow 2^R$ which means that the truster Tr authorizes the trustee to use some Tr's roles so that Te's issuer can assign Te's users to

these Tr's roles. For example: *TrustRole(EMS, SH)= neurologist, radiologist*, means that the tenant EMS exposes the two roles *neurologist* and *radiologist* to the tenant *SH* (as shown in figure 5).

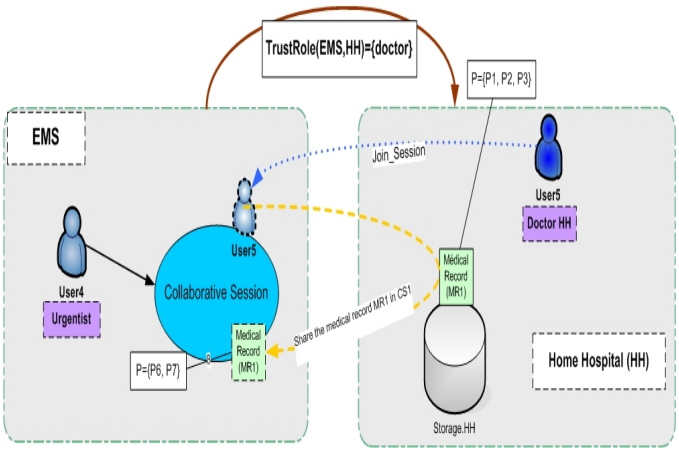

Figure. 5: The tenant trust role relation

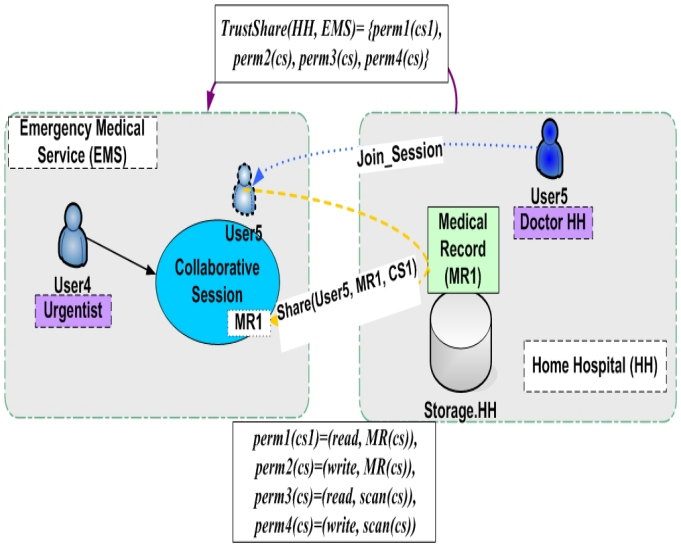

Figure. 6: The tenant trust share relation

B. Trust share relation

The tenant trust share relation $(TT) \subseteq T \times T$ is a many-to-many relationship between the truster Tr and the trustee Te. It is defined as $TrustShare(Tr, Te) = \cup_k(perm_k(cs)) = \cup_{ij}(a_i, objectType_j(cs))$, such as a_i is an action, $objectType_j$ is an object type defined by the tenant Tr and cs is a collaborative session. This relationship means that Tr's objects with the type $objectType_j$ could be shared in Te's collaborative sessions cs only to execute the action a_j. The trust is always established by the truster allowing the trustee to use truster's dynamic permissions. Therefore, the trustee can assign these permissions to the dynamic roles. For example as shown in figure 6, $TrustShare(HH, EMS) = perm1(cs), perm2(cs), perm3(cs), perm4(cs)$, such as:

- *perm1(cs)=(read, MR(cs))*
- *perm2(cs)=(write, MR(cs))*
- *perm3(cs)=(read, scan(cs))*
- *perm4(cs)=(write, scan(cs))*

means that the HH's all resources with the objects type MR and scan could be shared in EMS's collaborative sessions, respectively for the actions (read, write) and (read). The tenant EMS may assign these permissions to the roles in the session *doctor_EMS(cs)* and *neurologist(cs)*.

C. CTRBAC Model

A CTRBAC model has the following components:

- *I , T , U , R, P , S, CS, M, TA, TI, WS, WI, O, B* and *TT* are finite sets of *issuers, tenants, users, roles, permissions, sessions, collaborative sessions, collaborative sessions templates, tasks, task instances, workflows schema, workflows instances, objects, objects types* and tenant trust relation respectively;

- $TO \subseteq T \times I$, a many-to-one relation mapping each tenant to its owner issuer, $tenantOwner(t : T) \rightarrow I$, a function mapping a tenant to its owner issuer where $issuerOwner(t) = i \; iff \; (t, i) \in TO$;

- $MO \subseteq M \times T$, a many-to-one relation mapping each collaborative session template to its owner tenant, $CSTOwner(m : M) \rightarrow T$, a function mapping a collaborative session template to its owner tenant where $CSTOwner(m) = t \; iff \; (m, t) \in MO$

- $UO \subseteq U \times T$, a many-to-one relation mapping each user to its owner tenant, $userOwner(u : U) \rightarrow T$, a function mapping a user to its owner tenant where $userOwner(u) = t \; iff \; (u, t) \in UO$;

- $RO \subseteq R \times T$, a many-to-one relation mapping each role to its owner tenant, $roleOwner(r : R) \rightarrow T$, a function mapping a role to its owner tenant where $roleOwner(r) = t \; iff \; (r, t) \in RO$;

- $TAO \subseteq TA \times T$, a many-to-one relation mapping each task to its owner tenant, $TaskOwner(ta : TA) \rightarrow T$, a function mapping a task to its owner tenant where $TaskOwner(ta) = t \; iff \; (ta, t) \in TAO$;

- $WSO \subseteq WS \times T$, a many-to-one relation mapping each workflow schema to its owner tenant, $WorflowOwner(ws : WS) \rightarrow T$, a function mapping a worfkflow schema to its owner tenant where $WorkflowOwner(ws) = t \; iff \; (ws, t) \in WSO$;

- $PO \subseteq P \times T$, a many-to-one relation mapping each permission to its owner tenant, $permOwner(p : P) \rightarrow T$, a function mapping a permission to its owner tenant where $permOwner(p) = t \; iff \; (p, t) \in PO$;

- $OT \subseteq O \times T$, a many-to-one relation mapping each object to its owner tenant, $ObjectOwner(o : O) \rightarrow T$, a function mapping an object to its owner tenant where $ObjectOwner(o) = t \; iff \; (o, t) \in OO$;

- $BO \subseteq B \times T$, a many-to-one relation mapping each object type to its owner tenant, $ObjectTypeOwner(b : B) \rightarrow T$, a function mapping an object type to its owner tenant where $ObjectTypeOwner(b) = t\ iff\ (b,t) \in BO$;
- $OB \subseteq O \times B$ a many-to-one relation mapping each object to its type, $objectType(o : O) \rightarrow B$, a function mapping an object to its type where objectType $(o : O) = y$ iff $(o,b) \in OB$;
- $Instances(cst : CST) \rightarrow 2^{CS}$, a many-to-one relation mapping each collaborative session template to its collaborative session instances. For example, the relationship $Instance(NeuroEmergency) = \{cs1, cs2\}$ means that the collaborative sessions cs1 and cs2 are instances of the template neuroEmergency;
- $TaskInstances(ta : TA) \rightarrow 2^{TI}$, a many-to-one relation mapping each task to its task instances. For example, the relationship $TaskInstances(observation) = \{obs(cs1), obs(cs2)\}$ means that the task instances $obs(cs)$ and $obs(cs2)$ are instantiated from the task $observation$;
- $WorkflowInstances(ws : WS) \rightarrow 2^{wi}$, a many-to-one relation mapping each workflow schema to its workflow instances. For example, the relationship $workflowInstances(cliniccollaboration) = \{clinical(cs1), clinical(cs2)\}$ means that the workflow instances $clinical(cs)$ and $clinical(cs2)$ are instantiated from the workflow schema $cliniccollaboration$;
- $UA \subseteq U \times R$, a many-to-many user-to-role assignment relation requiring $(u,r) \in UA$ only if $userOwner(u) = roleOwner(r)$;
- $RTA \subseteq R \times TA$, a many-to-many role-to-task assignment relation requiring $(r,ta) \in RTA$ only if $roleOwner(r) = taskOwner(ta)$;
- $PTA \subseteq P \times TA$, a many-to-many permission-to-task assignment relation requiring $(p,ta) \in PTA$ only if $permOwner(p) = taskOwner(ta)$;
- $TAWS \subseteq TA \times WS$, a many-to-many task-to-workflow-schema assignment relation $requiring(ta,ws) \in TAWS$ only if $taskOwner(ta) = workflowOwner(ws)$;
- $Users(cs : C) \rightarrow 2^U$, a function mapping each collaborative session to a subset of users that are members of this session, $Users(cs) = \cup_{u \in U}\{u | Member(u,cs)\}$;
- $Sessions(cs : C) \rightarrow 2^S$, a function mapping each collaborative session to a subset of sessions, $Sessions(cs) = \cup_{s \in S}\{s | Member(user(s),cs)\}$;
- Role in the session $r(CS)$: is the role that will be activated when a user join a collaborative session. In the same session, we may have a set of members that play the same or different $roles : (u, r(cs)) \in UA$ if $(u,r) \in UA \wedge member(u,cs)$
- ObjectType in the session $ObjType(cs)$: are the objects with the type objType that are shared in the collaborative session $cs : obj \in objType(cs)$ if $(obj, objType) \in OB \wedge shared(obj,cs)$
- Permission in the session $P(CS)$: is the permissions that will be activated when a user joins a collaborative session, $p(cs) = (a, objType(cs)) : (u, p(cs)) \in UP$ if $(u, r(cs)) \subseteq UA \wedge (p(cs), r(cs), cst) \in PRA$
- $PTAM \subseteq P \times TA \times M$, a many-to-many-to-many permission-to-task-to-template assignment relation requiring $(p(cs), ta(cs), m) \in PTAM$ only if $permOwner(p) = taskOwner(ta) = templateOwner(m)$ and $cs \in instances(m)$;
- $\forall\ tij \in TI$ is an instance of task taj such as $tij \in taskInstances(taj)$; tij is an active task, denote $active(tij)$ if only if:
  - $taj \notin WFS$ or
  - $taj \in WFS \wedge (ta1, ta2, , tan \rightarrow taj) \wedge Completed(ti1, ti2, , tin)$
- $TrustRole(tr, te : T) \rightarrow 2^R$, a new function mapping a pair of truster and trustee tenants to a set of roles;
- $TrustShare(tr, te : T) \rightarrow 2^P$, a new function mapping a pair of truster and trustee tenants to a set of permissions related to the shared resources.

  Access control permissions are modeled by the following rule: $u \in U, p \in P, r \in R, cs \in CS, cst \in CST, ta \in TA, obj \in O, objType \in B$ such as: if
  $cs \in Instance(cst) \wedge$
  $(u, r(cs)) \in UA \wedge$
  $(r, ta) \in RTA \wedge$
  $(p(cs), ta(cs), cst) \in PTAM \wedge$
  $ta(cs) \in taskInstances(ta) \wedge$
  $Active(ta(cs))$
  $\rightarrow (u, p(cs)) \in UP$
  $p(cs) = (a, objType(cs))$
  $\rightarrow (u, (a, obj)) \in UP$
  - $[(u, r(cs)) \in UA if (u,r) \in UA \wedge member(u,cs)]$
  - $[obj \in objType(cs) if Type(obj, objType) \wedge Shared(obj,cs)]$

  which means that the user *u* has permission to perform the action *a* on the object *obj*, if an only if : (1) There is a collaborative session *cs* that is an instance of the template *cst*; (2) The user *u* plays the role *r* in the session *cs*; (3) The task *ta* is assigned to the role *r*; (4) The permission *p(cs)=(a, objType(cs))* is assigned to the task instance *ta(cs)* in the collaborative session template *cst*; (5) *ta(cs)* is an instance of the task *ta* ; (6) The task instance *ta(cs)* is active; (7) The object *obj* is of type *ObjType1* and is shared in the collaborative session *cs*. For instance, we define the security permission related to the rule: The *radiologist* member in the collaborative session of the template *NeuroEmergency* is authorized to read all the objects (of

the type scan: *scans_images*) shared in this collaborative session in the task *ta6(Interpret the scans images)*;
$cs1$ *is a collaborative session such as* :
$cs1 \in Instance(neuroEmergency) \wedge$
$(user1, radiologist(cs1)) \in UA \wedge$
$(radiologist, ta6) \in RTA \wedge$
$((read, scan(cs1)), ta6(cs1), neuroEmergency) \in PTAM \wedge$
$ta6(cs) \in taskInstances(ta6) \wedge$
$Active(ta6(cs))$
$\rightarrow (user1, (read, scan(cs1)) \in UP$
$[shared(scan1, cs1) \wedge scan1 \in scan(cs1)]$
$\rightarrow (user1, (read, scan1)) \in UP$

- *Active(ta6(cs1)) if Completed(ta1(cs1), ta2(cs1), ta3(cs1), ta4(cs1), ta5(cs1)*

V. ADMINISTRATIVE MODEL

In this section, we define the administrative model for the suggested approach. This model allows to the administrators to perform some administrative operations. Each administrative operation requires certain preconditions. In the following, we propose the formal specification of each administrative operation for a single issuer along with the corresponding preconditions:

- AssignUser(t, u, r)
  Precondition: $(t, i) \in TO \wedge (u, t) \in UO \wedge [(r, t) \in RO \cup tx \in T,\ r \in TrustRole(tx, t),\ where(r, tx) \in RO] \wedge (u, r) \notin UA$
- RevokeUser(t, u, r)
  Precondition: $(t, i) \in TO \wedge (u, t) \in UO \wedge (r, t) \in RO \cup tx \in T,\ r \in TrustRole(tx, t),\ where(r, tx) \in RO] \wedge (u, r) \in UA$
- AssignRoleTask(t, r, ta)
  Precondition : $(t, i) \in TO \wedge (ta, t) \in TAO \wedge [(r, t) \in RO \cup \forall tx \in T,\ r \in TrustRole(tx, t),\ where(r, tx) \in RO] \wedge (r, ta) \notin RTA$
- revokeRoleTask(t, r, ta)
  Precondition : $(t, i) \in TO \wedge (ta, t) \in TAO \wedge [(r, t) \in RO \cup \forall tx \in T,\ r \in TrustRole(tx, t),\ where(r, tx) \in RO] \wedge (r, ta) \in RTA$
- AssignPerm(t, ta, p)
  Precondition: $(t, i) \in TO \wedge (ta, t) \in TAO \wedge (p, t) \in PO \wedge (p, ta) \notin PTA$
- revokePerm(t, ta, p)
  Precondition: $(t, i) \in TO \wedge (ta, t) \in TAO \wedge (p, t) \in PO \wedge (p, ta) \in PTA$
- AssignPermCS(t, p(cs), ta(cs), m)
  Precondition: $(t, i) \in TO \wedge (m, t) \in MO \wedge \forall cs \in Instances(m) \wedge (ta, t) \in TAO \wedge ta(cs) \in taskInstances(ta) \wedge [(p(cs), t) \in PO \cup \forall ty \in T \wedge P(cs) \in TrustShare(ty, t)] \wedge (p(cs), ta(cs), m) \notin PTAM$
- RevokePermCS(t, p(cs), ta(cs), m)
  Precondition: $(t, i) \in TO \wedge (m, t) \in MO \wedge \forall cs \in Instances(m) \wedge (ta, t) \in TAO \wedge ta(cs) \in taskInstances(ta) \wedge [(p(cs), t) \in PO \cup \forall ty \in T \wedge P(cs) \in TrustShare(ty, t)] \wedge (p(cs), ta(cs), m) \in PTAM$
- AssignObject(t, obj, objType)
  Precondition: $(t, i) \in TO \wedge (obj, t) \in OO \wedge (obTypej, t) \in BO$
- addTenant(t)
  Precondition : $i \in I \wedge t \in T$
- deleteTenant(t)
  Precondition: $(t, i) \in TO \wedge t \in T$
- addCSTemplate(t,m)
  Precondition: $(t, i) \in TO$

VI. IMPLEMENTATION

For the implementation of our approach we used OpenStack Swift environment. It is a multi-tenant, highly scalable and durable software defined storage system designed to store files, videos, analytics data, web content, backups, images, virtual machine snapshots and other unstructured data. It allows building, operating, monitoring, and managing distributed object storage systems that can scale up to millions of users.
Swift enables users to store, retrieve, and delete objects (with their associated metadata) in containers via a RESTful HTTP API. Swift can be accessed with HTTP requests directly to the API or by using one of the many Swift client libraries such as Java, Python, Ruby, or JavaScript [16]. This makes it ideal as a primary storage system for data that needs to be stored and accessed via web based clients, devices and applications. The Account Server is responsible for listings of containers, while Container Server is responsible for listings of objects (Figure 7).
In our scenario, we consider we have three collaborating tenants: the school hospital tenant (SH), the emergency medical services tenant (EMS), and the home hospital tenant (HH). We specify the scenario within Swift component in the open source cloud-computing platform OpenStack [16]. We consider that each tenant is associated to an ACCOUNT (e.g. the accounts *ACC_SH, ACC_EMS* and *ACC_HH* represent the tenants *SH, EMS* and *HH* respectively).
A container is a mechanism that stores data objects. An account might have many containers, whereas a container name must be unique. This API enables a client to create a container, to set access controls and metadata, to retrieve a containers contents, and to delete a container.
A user is the entity that can perform actions on the object in the Account. Each user has its owner account and is associated to a single tenant. The account *ACC_SH* has four users:

- *ACC_user1*
- *ACC_user2*
- *ACC_user3*
- *ACC_user4*

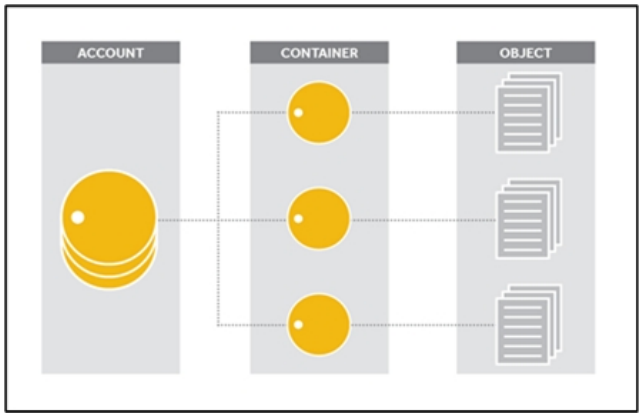

Figure. 7: Swift architecture[16]

In our example, each Account has many containers, and each container has a set of objects. For instance, the account ACC_HH has four containers: MR, DEC, SCAN, and DIAG. Swift uses the access control lists (ACL) to define the security permissions that regulate the usage of the container. Swift ACLs just assign permissions to users regarding the use of the objects of the account.

- *swift post -w '<ACL>' <container> [-A AUTH_URL] [-U user] [-K password]*
- *curl -X <PUT—POST> -i -H "X-Auth-Token: <TOKEN>" -H "X-Container-Write: <ACL>" <STORAGE_URL>/<container>*
- *curl -H "X-Auth-Token: <TOKEN>" -X PUT <STORAGE_URL>/<container>/<object> –data-binary @<filename>*

The collaborative applications requirements in Cloud environment are as follows:

- Swift ACLs support lists for read and write accesses. However, Swift does not allow specifying dynamic security rules defined in our approach such as: the User Role assignment, the permission role assignment and the relationships related to the collaboration sessions.
- It is necessary to have a component to manage collaborative sessions (Create collaborative sessions, Join/leave the session, Add users in sessions and share objects in the sessions).
- Trust relations among tenants, the truster exposes some trusters roles to the trustee, and this trustee assigns their users to these trusters roles so these users can access to the truster's resources by activating the trusters roles.
- It is necessary to have a component to support tasks and workflows. This component should enable the granting and revoking of permissions to be automated with the progression of the tasks in business process.

In order to support these requirements related to collaborative applications, we propose to add, in the swift environment, a new Policy module developed in shell language (figure 8). The policy module is composed of nine components: URT component, RT component, PA component, entities component, tasks component, object types component, session members component, tenant Trust Relations and shared objects components. In the following we describe each component of this module:

- URT component: in this component the security administrator defines the relationship between the tenant, the user and the role. *(ACC_SH: ACC_user1: neurologist)* means that in the tenant ACC_SH, the user ACC_user1 plays the role neurologist.
- RT component: in this component the security administrator defines the relationship between the tenant, the role and the task. For example, *(ACC_SH: neurologist: Take_decision)* means that in the tenant ACC_SH, the task Take_decision is assigned to the role neurologist.
- PA Component: In this component, the administrator can specify the permissions tasks assignment. In our scenario, we consider that the tenant EMS defines the policy rules. Note that at this level, we suppose that the security policy rules are valid and conflict-free. (A formal work is proposed by our team, based on finite-state machine in order to resolve conflicts problems detected in the security rules intra-tenants and cross-tenants).
- Entities Component: In this component, the administrator may define all the entities of the model (collaborative session templates, roles, users, objects and object types); and the relationships between these entities and their owner tenants.
- Object Types Component: In this component, the administrator may define all the relationships between objects and their types. These relationships are specified as follows: *(MR:mr1)*, which means that the object mr1 is of the type MR.
- Sessions members component: this component is responsible to manage users in the session such as Join/Leave collaborative sessions. The members of the collaborative sessions are specified in this component as follows: *(CS1: ACC_user1)*.
- Tasks component: In this component, the administrator specifies the relationship between tasks and the workflows schema can define the entities tasks and workflows schema. Moreover, the administrator defines the succession of tasks in the workflows schema. For example, *(ACC_SH: Take_decision: neurologydiagnosis)* means that in the tenant ACC_SH, the task Take_decision is assigned to the workflow neurologydiagnosis.
- Tenant Trust Relations: In this component, the administrator can specify the tenant trust relation established by the truster. These relationships are specified in this component as follows: *(ACC_EMS: ACC_SH: neurologist)*, which means that the tenant ACC_EMS authorizes the tenant ACC_SH to use the role neurologist. *(ACC_HH:ACC_EMS:perm1(cs))* means that the tenant ACC_HH authorizes the tenant ACC_EMS to use the permission perm1(cs).

- Shared objects Components: This component is responsible for the management of the shared resources in the session. Example: *(CS1:MR1)* means that the medical record MR1 is shared in CS1.

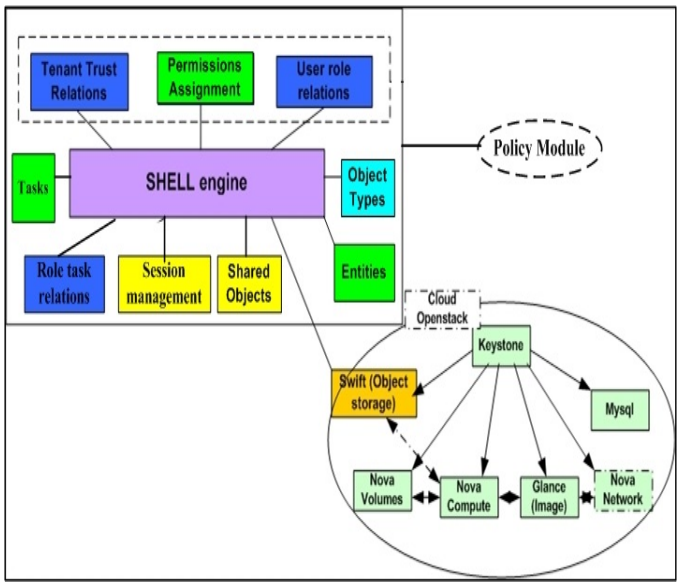

Figure. 8: Swift implementation

These components will be used by the Shell engine to evaluate the access request to resource in the collaborative session. When a user sends a request to access to a resource stored in the cloud swift, the Shell engine component evaluates this request according to the policy rules in order to decide whether the user is authorized to access to this resource or not. If the user is permitted to perform this action, then the policy module will execute an ACL command to assign this permission (read, medical_record) to the user in swift environment.

VII. CONCLUSION

In this paper, we formally define a new model CTRBAC that ensures access control to the shared resources within a collaborative session in multi-tenants environments, which means that collaborating users and the shared resources belong to the same or different tenants. In this model, we introduce new entities and trust relationships to support the sharing rules. These rules are used to secure resources sharing among users in collaborative sessions.
Furthermore, the suggested CTRBAC model introduces tasks and workflows in order enable the granting and revoking of permissions to be automated with the progression of the tasks in business process. Finally, we demonstrated the feasibility of our suggestion by an implementation in the SwiftStack environment. As a future work, we plan to extend this approach to ensure access control during collaborative sessions in heterogeneous multi-clouds environments.

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] R. Sandhu, E. J. Coyne, H. L. Feinstein and C.E.Youman, Role-based access control models, IEEE Computer, 29(2):38-47, 1996.

[3] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn and R. Chandramouli, Proposed NIST Standard for Role-Based Access Control, ACM Transactions on Information and System Security, 4(3), pp. 222-274, AUGUST 2001.

[4] S. I. Gavrila and J. F. Barkley, Formal specification for Role Based Access Control User/Role and Role/Role Relationship Management, Third ACM Workshop on Role-Based Access Control, pp. 81-90, October 1996.

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

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

[7] 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. 4855, Nov/Dec 2010.

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

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

[10] A. Almutairi, M. Sarfraz, S. Basalamah, W. Aref, A Ghafoor, A Distributed Access Control Architecture for Cloud Computing, IEEE Software 29(2): 36-44, 2012.

[11] M. Singhal, S. Chandrasekhar, T. Ge, R. Sandhu, R. Krishnan, E. Bertino, Collaboration in Multicloud Computing Environments: Framework and Security Issues, IEEE Computer 46(2): 76-84, 2013.

[12] F. Cuppens, Cuppens-Boulahia, N. B. Talbi, M. Morucci, S. Essaouni, Smatch: Formal dynamic session management model for RBAC, J. Inf. Sec. Appl. 18 (1) , 30-44, 2013.

[13] H. Takabi, J. B. D. Joshi, and G. J. Ahn, Security and Privacy Challenges in Cloud Computing Environments, IEEE Security and Privacy, Vol. 8, No. 6, pp. 25-31, 2010.

[14] 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.

[15] 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).

[16] OpenStack Swift Architecture, https://swiftstack.com/openstack-swift/architecture/.

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

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

[19] Y.Zhang, J.Joshi, Access Control and Trust Management for Emerging Multidomain En-vironments, in Annals of Emerging Research in Information Assurance, Security and Pri-vacy Services, Editors: S.Upadhyaya, R. O. Rao 2009.

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

[21] 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