A Multi-Tenant RBAC Model for Collaborative Cloud Services

Bo Tang*†, Qi Li* and Ravi Sandhu*†
*Institute for Cyber Security
†Department of Computer Science
University of Texas at San Antonio
One UTSA Circle, San Antonio, Texas 78249
Email: btang@cs.utsa.edu, {qi.li2, ravi.sandhu}@utsa.edu

***Abstract*—Most cloud services are built with multi-tenancy which enables data and configuration segregation upon shared infrastructures. In this setting, a tenant temporarily uses a piece of virtually dedicated software, platform, or infrastructure. To fully benefit from the cloud, tenants are seeking to build controlled and secure collaboration with each other. In this paper, we propose a Multi-Tenant Role-Based Access Control (MT-RBAC) model family which aims to provide fine-grained authorization in collaborative cloud environments by building trust relations among tenants. With an established trust relation in MT-RBAC, the trustee can precisely authorize cross-tenant accesses to the truster's resources consistent with constraints over the trust relation and other components designated by the truster. The users in the trustee may restrictively inherit permissions from the truster so that multi-tenant collaboration is securely enabled. Using SUN's XACML library, we prototype MT-RBAC models on a novel Authorization as a Service (AaaS) platform with the Joyent commercial cloud system. The performance and scalability metrics are evaluated with respect to an open source cloud storage system. The results show that our prototype incurs only 0.016 second authorization delay for end users on average and is scalable in cloud environments.**

***Keywords*—cloud computing; multi-tenancy; trust; collaboration; fine-grained authorization**

I. Introduction

The growing predominance of cloud computing impacts every aspect of the information technology (IT) industry [1]. It brings business agility and lower costs for information systems by using virtualization and shared infrastructures. Most cloud providers isolate tenants from each other using multi-tenancy [2] to secure user data and configurations. However, the isolation strategy hampers multi-tenant collaborations which are also essential in the cloud [3]. In particular, fine-grained secure resource sharing among tenants is not fostered in today's commercial clouds [4], [5].

Utilizing existing access control models, cloud providers are capable to control user activities within a single tenant. For example, NASA integrates Role-Based Access Control (RBAC) in the cloud to enforce fine-grained access control in their existing directories [6]. Nevertheless, the traditional RBAC model [7] is not designed to enforce collaborative access control in decentralized environments. Currently, database schema is widely utilized to enable multi-tenant data sharing for Software as a Service (SaaS) [8], [9]. However, this approach is confined to address multi-tenant access control in databases and cannot be directly extended to protect other vital types of resources such as files and virtual machines. Consequently, in order to enable secure multi-tenant collaborations in the cloud, we need a general fine-grained access control model for this purpose.

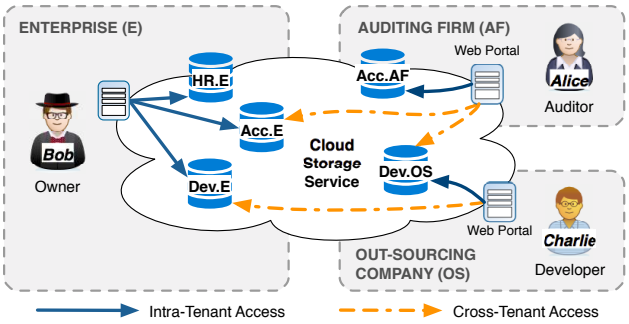


Fig. 1: Multi-tenant accesses in an out-sourcing case

To motivate the problem, we use an out-sourcing example illustrated in Figure 1, in which the *Enterprise* (E), the *Out-Sourcing company* (OS), and the *Auditing Firm* (AF) are three collaborating parties sharing a common cloud storage service. E out-sources part of its application development work to OS and external auditing tasks to AF. The cloud storage service provides storage services for the development department of OS, the accounting department of AF, and three of E's departments, development, accounting, and HR, as segregated tenants. Let "." denote the affiliation relation between the tenant and its organization (also called issuer) so that, for example, $Dev.E$ represents the tenant corresponding to E's development department. The example cross-tenant accesses for collaborations are as follows.

C1. *Charlie* as a *developer* in OS has to access the source code stored in $Dev.E$ to perform his out-sourcing job;
C2. *Alice* as an *auditor* in AF requires read-only access to financial reports stored in $Acc.E$; and
C3. *Alice* needs read-only accesses to $Dev.E$ and $Dev.OS$ in order to audit the out-sourcing project.

For simplicity, in our examples we assume all the tenants are created on a single cloud service, bringing homogeneous architecture which is often the case in cloud environments [10]. However, we do not exclude the potential of heterogeneous collaborations among multiple cloud services or even among multiple service models: SaaS, Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [2]. As a common collaboration need suggests, a user may test and deploy the source code stored in $Dev.E$ directly on another tenant of a PaaS service. This function requires secure collaborative accesses

978-1-4673-5839-2/13/$31.00 ©2013 IEEE

between the two services. In fact, SaaS services are often hosted on PaaS clouds in which the SaaS services are regarded as tenants. The similar situation holds between PaaS and IaaS. Thus, we treat accesses among multiple tenants, clouds, or even service models equivalently in the abstraction level as multi-tenant accesses upon which access control mechanisms should be enforced.

In order to achieve multi-tenant access control, Calero et al [11] propose a multi-tenancy authorization system (MTAS) by extending RBAC with a coarse-grained trust relation. Multi-tenant collaborations are enabled in MTAS by bridging two tenants with a cross-tenant trust relation. The tenant establishing the trust is called the truster and the tenant being trusted is called the trustee. The essence of collaboration is resource sharing wherein a resource requester requests to share resources from a resource owner. The resources here represent the objects, such as virtual machine instances or stored files, in a tenant leased from the cloud service provider (CSP). For instance, in order to enable cross-tenant access C1 in the out-sourcing example above, an MTAS trust relation is established with $Dev.OS$ as truster and $Dev.E$ as trustee, allowing $Dev.E$ as resource owner to issue suitable cross-tenant assignments to allow $Dev.OS$'s users to access $Dev.E$'s resources. By definition, the MTAS trust relation is always established by the resource requester. The granularity of MTAS trust is refined in [12] to reduce undue exposure of the truster's sensitive authorization information to the trustees. While Calero et al provide use cases for this resource-requester initiated cross-tenant trust relation, there are obviously other practically useful approaches that can be developed.

In this paper we develop a family of models for a cross-tenant trust relation that is established by the resource owner. We demonstrate the utility of this approach by means of use cases. Later in the paper we formally compare the role-based trust models of MTAS, the current paper and RT [13]. There may be possibility of additional cross-tenant trust models, based on roles and perhaps on attributes. Eventually we believe there will be consolidation and unification of cross-tenant trust models but we are currently at early stages of developing and investigating alternate cross-tenant trust models.

Our central contribution is a novel family of Multi-Tenant RBAC (MT-RBAC) models where the cross-tenant trust relation is established by the resource owner rather than by the resource requester. MT-RBAC extends the traditional RBAC model [7] with two new built-in entity components: issuers and tenants.[1] Each issuer can have multiple tenants in the cloud, whereas each tenant belongs to a single issuer. A cross-tenant trust relation is established and maintained by the issuer of the resource owner tenant (i.e., truster), as opposed to the resource requester tenant (i.e., trustee).

Three MT-RBAC models integrate three different trust relations with increasingly finer-grained constraints, respectively tenant trust (MT-RBAC$_0$), trustee independent public roles (MT-RBAC$_1$), and trustee dependent public roles (MT-RBAC$_2$). To allow the trustee to access the truster's resources, the base model MT-RBAC$_0$ requires the truster to expose its entire role set and corresponding authorization assignments to the trustee. For example, to achieve the cross-tenant access C1 in the out-sourcing example, E will assign a trust relation from $Dev.E$ to $Dev.OS$ so that OS can use E's authorization information to issue appropriate cross-tenant assignments enabling $Dev.OS$'s users to access $Dev.E$'s resources. To limit unnecessary exposure of the truster's authorization information, MT-RBAC$_1$ requires only exposing the truster's public roles to all the trustees, these being the same for all trustees. Further, MT-RBAC$_2$ enforces finer-grained constraints by exposing the truster's public roles on a trustee by trustee basis. With these trust relations, a truster's issuer can flexibly and efficiently constrain accesses from the corresponding trustees in a suitably fine-grained manner.

We demonstrate the feasibility of MT-RBAC by prototyping it with SUN's XACML library [14]. We develop and deploy a novel Authorization as a Service (AaaS) platform applying MT-RBAC models in a Joyent cloud [15]. We systematically evaluate the performance of MT-RBAC with an open source cloud storage service [16]. The experimental results show the MT-RBAC policies with different expressive power in the AaaS service incur various policy evaluation delays. Evaluation of MT-RBAC policies takes on average no more than 0.016 second overhead in downloading files. Therefore, we believe that the performance of the prototype system is acceptable for cloud services. Further, we observe that the prototype system is also scalable in cloud settings.

The rest of the paper is organized as follows. Section II formally presents the MT-RBAC model family along with the introduction of its administrative model AMT-RBAC and constraints to be considered. Section III describes the prototype implementation and evaluation results. Section IV presents related work and compares our work with others in terms of role-based trust models. Section V concludes the paper.

II. A Family of MT-RBAC Models

To achieve fine-grained access control for multi-tenant collaborations in the cloud, we develop a family of three MT-RBAC models with increasingly finer-grained trust relations.

A. Overview

MT-RBAC models, as shown in Figure 2, have six entity components: *issuers* (I), *tenants* (T), *users* (U), *permissions* (P), *roles* (R) and *sessions* (S). The traditional RBAC [7] entities of users, permissions and roles now have a tenant attribute so that they can be uniquely identified as depicted by the user-ownership (UO), permission-ownership (PO) and role-ownership (RO) relations respectively in Figure 2. All three relations are many-to-one relations from users, permissions and roles respectively to their owner tenants. Further, another many-to-one relation representing tenant-ownership (TO) exists between tenants and issuers.

ISSUERS. An issuer is a client of a single or multiple cloud services. Typically, it is either an organization or an individual who is able to administer its own tenants in the cloud services. For simplicity, we consider a single cloud scenario in this paper, so the name of the cloud service is not explicitly specified. For instance, in the out-sourcing example,

[1] MTAS [11] does not distinguish issuers and tenants, which are regarded as equivalent notions in that model. This distinction is important for our purpose in this paper, particularly for elaboration of the administrative model.

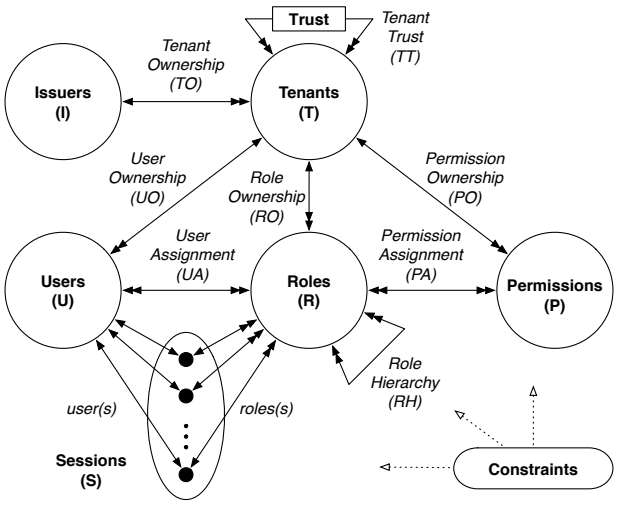


Fig. 2: MT-RBAC Model

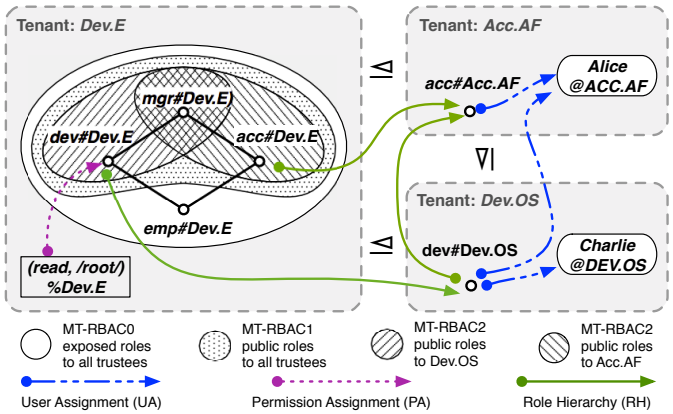


Fig. 3: Example multi-tenant assignments in the out-sourcing case. The differences among the MT-RBAC models are also illustrated in terms of E's exposed roles.

E, OS and AF represent three issuers respectively in a single cloud storage service.

TENANTS. A tenant is an exclusive virtual partition of a cloud service leased from a cloud service provider. An issuer may own multiple tenants while a tenant belongs to a single issuer. Let "." denote the tenant-issuer relation. For example, $Dev.OS$ represents the tenant Dev of the issuer OS.[2]

USERS. A user is an identifier for an individual person associated with a single tenant. An individual can act as different users in different tenants. Let "@" denote the user-tenant relation. For example, $Alice@Acc.AF$ and $Alice@Acc.E$ are two different users in tenants $Acc.AF$ and $Acc.E$ respectively, even if they belong to a single person, Alice.[3]

ROLES. A role is a job function (role name) associated with a tenant. A role belongs to a single tenant while a tenant may own multiple roles. Let "#" denote the role-tenant relation $roleName\#tenant$. For example, $dev\#Dev.E$ represents a developer role in tenant $Dev.E$.

PERMISSIONS. A permission is a specification of a privilege to an object in a tenant. A permission is associated with a single tenant while a tenant may have multiple permissions. Let "%" denote the permission-tenant relation $(privilege, object)\%tenant$. For example, $(read, /root/)\%Dev.E$ represents a permission to read the "/root/" path on $Dev.E$.

SESSIONS. A session is an instance of activity established by a user. A subset of roles that the user is assigned to can be activated in a session. Note that in multi-tenant cloud environments the user and the active roles of a session are not necessarily from a single tenant.

Crucially, in order to address collaborations among tenants, MT-RBAC introduces a role-based trust relation, Tenant Trust (TT), as illustrated in Figure 2.

TT is reflexive but not transitive, symmetric or anti-symmetric. In MT-RBAC, a trust relation is always established by the truster's issuer. It enables the trustee's issuer to add trustee's users to truster's roles. This can be done directly on a per user basis by assigning one of B's (trustee's) users to one of A's (truster's) roles via UA. Alternately it can be done indirectly by assigning one of B's roles, say r_1, to be senior to one of A's roles, say r_2, via RH. Thereby all members of r_1 become members of r_2 and A's permissions associated with r_2 are granted to the B's users in r_1. We emphasize that trust is established at the granularity of tenants. For example, if AF asserts $Acc.AF \trianglelefteq Acc.E$, there is no trust from $Acc.AF$ to $HR.E$ or $Dev.E$. The formalization of TT and its effects are described in Definition 1 as follows.

Definition 1: The tenant trust relation $TT \subseteq T \times T$ is a many-to-many reflexive relation on T, also written as "$\trianglelefteq$". By asserting $A \trianglelefteq B$, A's issuer exposes A's roles to B's issuer so that B's issuer can and can only make the following two kinds of assignments:

- assigning B's users to A's roles; and
- assigning A's roles as junior roles to B's roles.

Different MT-RBAC models vary in the granularity of the trust relations, specifically in the truster's role exposure. The left part of Figure 3 shows the role hierarchy of $Dev.E$ in the out-sourcing example. Since MT-RBAC$_0$ does not enforce any constraint on the trust relation, the entire $Dev.E$ role structure is exposed to all $Dev.E$'s trustees (more precisely to their issuers). In MT-RBAC$_1$, suppose the employee role $emp\#Dev.E$ is a private role while the others are public. The private role is never exposed to other tenants. Conversely, the public roles are exposed to all the trustees' issuers. Crucially, during cross-tenant accesses, the permissions associated with the private role cannot be inherited directly or even indirectly through the public roles. In MT-RBAC$_2$, public role sets are customized for different trustees. Suppose the accountant role $acc\#Dev.E$ and the manager role $mgr\#Dev.E$ have to be exposed in the collaboration with $Acc.AF$ while $mgr\#Dev.E$ and the developer role $dev\#Dev.E$ need to be exposed to

[2]In a more general treatment we would identify the cloud service explicitly in a three part name such as $Dev.OS.CloudService$.

[3]Other user models are possible. For instance, the users of a same person can be combined into one universal ID using federated identity [17]–[19]. However, since this mechanism is not fully supported in contemporary clouds, MT-RBAC models do not require a universal ID for an individual. The particular user model chosen does not materially impact the essentials of MT-RBAC. For completeness we do need a concrete user model.

$Dev.OS$ for the out-sourcing project. Accordingly, as illustrated in Figure 3, there are two different public role sets to these two trustees respectively. Note that even if $mgr\#Dev.E$ is public in both collaborations, for either one, it only inherits permissions from the junior roles in the corresponding public role set. For example, since $acc\#Dev.E$ is a private role to $Dev.OS$ and a public role to $Acc.AF$, its permissions are only inherited to $Acc.AF$ but not to $Dev.OS$.

B. Base Model—MT-RBAC$_0$

In order to enable secure cross-tenant collaborations, the base model is formally defined as follows.

Definition 2: The Base Model MT-RBAC$_0$ has the following components.

- I, T, U, R, P, S and TT are finite sets of issuers, tenants, users, roles, permissions, sessions and tenant trust relation respectively;
- $TO \subseteq T \times I$, a many-to-one relation mapping each tenant to its owner issuer, also written as ":"; correspondingly, $tenantOwner(t : T) \rightarrow I$, a function mapping a tenant to its owner issuer where $userOwner(t) = i$ iff $(t, i) \in TO$;
- $UO \subseteq U \times T$, a many-to-one relation mapping each user to its owner tenant, also written as "@"; correspondingly, $userOwner(u : U) \rightarrow T$, 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, also written as "#"; correspondingly, $roleOwner(r : R) \rightarrow T$, a function mapping a role to its owner tenant where $roleOwner(r) = t$ iff $(r, t) \in RO$;
- $PO \subseteq P \times T$, a many-to-one relation mapping each permission to its owner tenant, also written as "%"; correspondingly, $permOwner(p : P) \rightarrow T$, a function mapping a permission to its owner tenant where $permOwner(p) = t$ iff $(p, t) \in PO$;
- $canUse(r : R) \rightarrow 2^T$, a function mapping a role to a set of tenants who can use the role; formally, $canUse(r) = \{t \in T | roleOwner(r) \trianglelefteq t\}$;
- $UA \subseteq U \times R$, a many-to-many user-to-role assignment relation requiring $(u, r) \in UA$ only if $userOwner(u) \in canUse(r)$;
- $PA \subseteq P \times R$, a many-to-many permission-to-role assignment relation requiring $(p, r) \in PA$ only if $permOwner(p) = roleOwner(r)$;
- $RH \subseteq R \times R$ is a partial order on R called role hierarchy or role dominance relation, also written as "$\geq$", requiring $r_2 \geq r_1$ only if $roleOwner(r_2) \in canUse(r_1)$;
- $user(s : S) \rightarrow U$, a function mapping each session to a single user which is constant within the life-time of the session; and
- $roles(s : S) \rightarrow 2^R$, a function mapping each session to a subset of roles, $roles(s) \subseteq \{r | \exists r_2 \geq r[(user(s), r_2) \in UA \wedge userOwner(user(s)) \in canUse(r)]\}$, which can change with time, and s has the permissions $\bigcup_{r \in roles(s)} \{p | (p, r) \in PA\}$.

Note that the introduction of the derived $canUse$ function provides convenient means for TT to take effect upon UA and RH. For a given role $r \in R$, the statement $roleOwner(r) \in canUse(r)$ is always true since TT is reflexive. Hence, intra-tenant assignments are always under the authority of the tenant's owner issuer.

A trustee (more precisely its issuer) can assign a truster's permissions to the trustee's users only at the granularity of the truster's roles. In the spirit of RBAC, MT-RBAC does not allow individual permissions of the truster to be assigned to the trustee's users or roles.

Role activation mechanisms determine the executable permissions in a session. Because a role may inherit permissions from its junior roles in the role hierarchy, when a role is activated in a session, its inherited roles may be either automatically activated (implicit activation) or require explicit activation. The choice between the two approaches is left as an implementation issue in the NIST RBAC model [7]. In the RBAC96 model implicit activation is specified [20]. In MT-RBAC, we choose to specify explicit activation in the $roles(s)$ component. In a session, only the permissions of the explicitly activated roles are available. An alternate forumulation of MT-RBAC with implicit activation can also be developed.

The revocation of a TT relation $t_r \trianglelefteq t_e$ can be asserted by t_r's issuer. This operation will automatically eliminate the trustee t_e from $canUse(r)$ for each of $\{r \in R | (r, t_r) \in RO\}$. Moreover, as the formal description of UA and RH in Definition 2 suggests, all the relevant cross-tenant assignments (i.e., UA and RH) issued by t_e's issuer will be revoked automatically, as well as the active roles in the sessions of t_e's users. In this way the permissions in t_r are not able to be inherited to t_e after revocation.[4] If the removed trust relation is subsequently restored, the trustee's issuer would have to redefine and reissue all the cross-tenant assignments from scratch.

MT-RBAC$_0$ enables multi-tenant collaborations by means of TT. However the coarse-grained trust relation may lead to breaches in protection of sensitive information. For example, in the base model, a truster needs to expose all the organization role structure to its trustees. A more important issue is that trustees can obtain more sensitive information by assigning their users to the sensitive roles they can use. Therefore, MT-RBAC$_0$ may only be suitable for collaborations among closely related tenants such as departments of a single organization.

C. Trustee Independent Public Role—MT-RBAC$_1$

A natural enhancement to address the granularity limitations of the base model is to classify the components for collaborations into public ones and private ones. In this setting, collaborations only take place on the public components of the resource owner. The truster's roles can be simply classified into two disjoint sets: *public roles* and *private roles*. Since the truster's public roles are public equally to all the trustees, we name this mechanism as trustee independent public role

[4]For simplicity and security in model level, the corresponding cross-tenant assignments issued by trustee's issuer are automatically cleared as soon as the trust relation is revoked. Depending upon implementation, trustee's issuer may also choose to manually clear or even keep the nonfunctional hanging cross-tenant assignments for future use although it is not recommended.

($TIPR$) which provides more expressiveness and granularity than TT in MT-RBAC$_0$.

Definition 3: MT-RBAC$_1$ inherits all the components from MT-RBAC$_0$ as described in Definition 2, while the following modifications are applied.

- $\mathcal{P}_{TI}(t : T) \rightarrow 2^R$, a new function mapping a tenant to a set of roles which is the trustee independent public role ($TIPR$) set of the tenant; and
- $canUse(r : R) \rightarrow 2^T$ is modified to $canUse(r) = \{t\} \cup \{t_e \in T | t \trianglelefteq t_e \wedge r \in \mathcal{P}_{TI}(t)\}$, where $(r, t) \in RO$.

Note that the truster's permissions associated with the public roles can be inherited externally to the trustees, but those associated with the private roles can only be inherited internally within the truster. For example, in Figure 3, since the employee role $emp\#Dev.E$ is a private role, its permissions should never be used in cross-tenant accesses. Further, if $\mathcal{P}_{TI}(t)$ consists of the entire role set of t, then MT-RBAC$_1$ is identical to MT-RBAC$_0$. Therefore, MT-RBAC$_1$ is a more general model than the base model.

MT-RBAC$_1$ provides enhanced security by introducing $TIPR$ so that only public roles are exposed to the trustees. Besides, it enables finer-grained control for the resource owner's issuer, say i_r, who can revoke a specific cross-tenant access from a trustee, say t_e by removing the relevant roles from $\mathcal{P}_{TI}(t_r)$ while not disrupting the other cross-tenant accesses from t_e. A public role should be automatically removed from $\mathcal{P}_{TI}(t_r)$ as soon as the corresponding trust relation with t_r comes to an end. However, in practice a trustee independent public role may be used in multiple trust relations so that the bindings of the public role and its corresponding trust relations should be carefully managed; otherwise the public role may be unnecessarily maintained or unwillingly removed. Therefore, MT-RBAC$_1$ is only suitable for a single type or very similar types of collaborations.

In MT-RBAC$_1$, the $\mathcal{P}_{TI}(t)$ function lacks the expressiveness to describe various public role sets for different trustees. To achieve the least privilege principle in multi-tenant collaborations, we propose MT-RBAC$_2$ which is the most fine-grained model in the MT-RBAC family.

D. Trustee Dependent Public Role—MT-RBAC$_2$

Unlike MT-RBAC$_1$ treating all the trustees equally, MT-RBAC$_2$ supports different public role sets specifically for different trustees. With respect to every established trust relation, the truster's issuer can designate disjoint sets of public roles and private roles with respect to the trustee. This mechanism is called trustee dependent public role ($TDPR$) which provides more expressiveness and flexibility for the truster's issuer to maintain cross-tenant accesses for different trustees. The formal definition of MT-RBAC$_2$ follows.

Definition 4: MT-RBAC$_2$ inherits all the components from MT-RBAC$_0$ as described in Definition 2, while the following modifications are applied.

- $\mathcal{P}_{TD}(t_r, t_e : T) \rightarrow 2^R$, a new function mapping a pair of truster and trustee tenants to a set of roles which is the trustee dependent public role ($TDPR$) set of the truster to the trustee; and
- $canUse(r : R) \rightarrow 2^T$ is modified to $canUse(r) = \{t\} \cup \{t_e \in T | t \trianglelefteq t_e \wedge r \in \mathcal{P}_{TD}(t, t_e)\}$, where $(r, t) \in RO$.

Note that if for every trustee t_e the $\mathcal{P}_{TD}(t_r, t_e)$ function returns the same set of public roles, then MT-RBAC$_2$ is equivalent to MT-RBAC$_1$. Thus, MT-RBAC$_2$ is more general than MT-RBAC$_1$ and the most general model in the MT-RBAC model family.

Comparing to MT-RBAC$_1$, the revocation process in MT-RBAC$_2$ is much simpler. The revocation of a cross-tenant access can be easily achieved by removing the relevant roles from the specific $TDPR$ set. This operation is executed by the truster's issuer and will not affect other accesses from other trustees.

MT-RBAC$_2$ supports various types of collaborations since $TDPR$ sets are maintained per truster-trustee pair. The truster's issuer has to maintain a $TDPR$ set for each trustee.

E. Administrative MT-RBAC (AMT-RBAC) model

The administration of tenant trust relations and authorization assignments is critical in MT-RBAC. Since TT is embedded in the cross-tenant assignments (i.e., UA and RH) as described in Definition 2, the management of tenant trust relations also controls cross-tenant accesses.

The core idea of the administrative model for MT-RBAC is dual control, meaning both of the truster's and the trustee's issuers have complementary power of authority to control cross-tenant accesses. The cross-tenant assignments are created and maintained by the trustee's issuer. The effectiveness of cross-tenant assignments depends on the corresponding trust relations which are under the control of the truster's issuer. In this way, the security and efficiency of the administration process are convenient for both parties. The truster's issuer deals with the overall trust and constraints for a trustee. The trustee's issuer deals with the finer details of the trustee's users.

Definition 5: The Administrative MT-RBAC (AMT-RBAC) model requires both issuers of the collaborating tenants, i_A the resource owner A's issuer and i_B the requester B's issuer, to manage the tenant trust relations and the cross-tenant authorization assignments separately as follows.

- i_A is responsible for managing the tenant trust relation of $A \trianglelefteq B$; and
- i_B is responsible for managing the cross-tenant assignments (i.e., UA and RH) to A's roles, according to Definition 2.

Note that the resource owner's issuer delegates the management of the cross-tenant authorization assignments to the resource requester's issuer. With the carefully defined cooperative mechanisms, in AMT-RBAC the resource owner's issuer retains the critical management authority (managing trust relations) while the maintenance of cross-tenant assignments is given away to the resource requester's issuer who is more knowledgable of the requesting users and roles.

In Table I, we give the formal specification of the exact administration functions of AMT-RBAC for a single issuer i along with the corresponding preconditions and updates to MT-RBAC policies.

TABLE I: Administration functions available to issuer *i* in AMT-RBAC

Function	Precondition	Update
$assignUser(t, r, u)$	$(t, i) \in TO \wedge (u, t) \in UO \wedge t \in canUse(r)$	$UA' = UA \cup \{(u, r)\}$
$revokeUser(t, r, u)$	$(t, i) \in TO \wedge (u, t) \in UO \wedge t \in canUse(r) \wedge$ $(u, r) \in UA$	$UA' = UA \setminus \{(u, r)\}$
$assignPerm(t, r, p)$	$(t, i) \in TO \wedge (r, t) \in RO \wedge (p, t) \in PO$	$PA' = PA \cup \{(p, r)\}$
$revokePerm(t, r, p)$	$(t, i) \in TO \wedge (r, t) \in RO \wedge (p, t) \in PO \wedge$ $(p, r) \in PA$	$PA' = PA \setminus \{(p, r)\}$
$assignRH(t, r_{asc}, r_{desc})$	$(t, i) \in TO \wedge (r_{asc}, t) \in RO \wedge t \in canUse(r_{desc}) \wedge$ $\neg(r_{asc} \gg r_{desc})$ † $\wedge \neg(r_{desc} \geq r_{asc})$ ‡	$\geq' = \geq \cup \{r, q : R \mid r \geq r_{asc} \wedge r_{desc} \geq q \wedge$ $roleOwner(r) \in canUse(q) \bullet (r, q)\}$
$revokeRH(t, r_{asc}, r_{desc})$	$(t, i) \in TO \wedge (r_{asc}, t) \in RO \wedge t \in canUse(r_{desc}) \wedge$ $r_{asc} \gg r_{desc}$	$\geq' = (\gg \setminus \{(r_{asc}, r_{desc})\})^*$ §
$assignTrust(t, t_1)$	$t_1 \in T$	$\trianglelefteq' = \trianglelefteq \cup \{(t, t_1)\}$
$revokeTrust(t, t_1)$	$t_1 \in T \wedge t \neq t_1 \wedge t \trianglelefteq t_1$	$\trianglelefteq' = \trianglelefteq \setminus \{(t, t_1)\}$ ¶
$addTenant(t)$	$i \in I \wedge t \notin T$	$T' = T \cup \{t\}$
$deleteTenant(t)$	$(t, i) \in TO \wedge t \in T$	$[\forall t_1 : T \Rightarrow revokeTrust(t, t_1)]$ $[\forall t_2 : T \Rightarrow revokeTrust(t_2, t)]$ $UA' = UA \setminus \{(u, r) \mid (u, t) \in UO \wedge (r, t) \in RO\}$ $PA' = PA \setminus \{(p, r) \mid (p, t) \in PO \wedge (r, t) \in RO\}$ $RH' = RH \setminus \{(r, r') \mid (r, t) \in RO \wedge (r', t) \in RO\}$ $U' = U \setminus \{u \mid (u, t) \in UO\}$ $UO' = UO \setminus \{(u, t) \mid u \notin U\}$ $R' = R \setminus \{r \mid (r, t) \in RO\}$ $RO' = RO \setminus \{(r, t) \mid r \notin R\}$ $P' = P \setminus \{p \mid (p, t) \in PO\}$ $PO' = PO \setminus \{(p, t) \mid p \notin P\}$ $T' = T \setminus \{t\}$ $TO' = TO \setminus \{(t, i)\}$

† The notation "$\gg$" represents an immediate inheritance relation. For example, $r_{asc} \gg r_{desc}$ means that r_{asc} is a parent of r_{desc}.
‡ This condition avoids the creation of role cycles which is discussed in Section II-F.
§ The notation "$*$" represents recursive updates for the entire RH assignments. Implied RH relations are preserved after revocation.
¶ The revocation of a trust relation automatically triggers updates in the $canUse()$ function of all t's roles and then corresponding UA and RH accordingly.

Note that both of the $assignTrust$ and $revokeTrust$ functions result in automatic updates of the $canUse$ function for each of the truster's roles. Moreover, since the public role sets in MT-RBAC$_1$ and MT-RBAC$_2$ also can be modified by the truster's issuer, the return sets of $canUse(r)$ for the truster's roles are updated accordingly. Because $canUse$ function is updated, the trustee's cross-tenant assignments, UA and RH, and their authorized cross-tenant accesses are also updated accordingly. In this way, the cross-tenant assignments are not only controlled by the trustee's issuer, but also manageable by the truster's issuer.

F. Constraints

The constraints identified in RBAC96 [20] are directly applicable to MT-RBAC for intra-tenant accesses. While the trust relations in MT-RBAC enable cross-tenant accesses, constraints should also be extended into a multi-tenant environment. Some constraints we feel reasonable to implement in the new environment are described below.

Role Cycles. A "role cycle" may be formed across tenants in MT-RBAC systems without proper constraints. This is a well known problem in inter-domain access control [21]. The issue may lead to implicit role upgrades in the role hierarchy. A role r_{1a} in tenant A may be assigned as a senior role to r_{2b} in tenant B while r_{2b} dominates r_{1b} in B. If r_{1b} is then assigned as a senior role to r_{2a} which dominates r_{1a} in A, then a role cycle forms. As a result, a user in r_{1a} can inherit permissions from a senior role r_{2a} through the cross-tenant RH assignments contrary to the role hierarchy in A. In order to prevent the formation of role cycles, constraints should be enforced over assignments or sessions. Constraints with respect to assignments can be enforced by checking for role cycles whenever a cross-tenant RH assignment is issued. Constraints with respect to sessions would allow role-cycles in assignments, but prohibit a cycle of roles from being activated in a single session. The algorithms for checking role cycles are well-known and straightforward.

Separation of Duties. There are two levels of separation of duty (*SoD*) problem that we need to be concerned during collaboration in MT-RBAC systems. These are at the tenant level and role level. For tenant level *SoD*, one collaborating tenant cannot execute two conflicting responsibilities. For instance, SOX [22] compliant companies are not suppose to hire one third-party as both consultant and auditing organization. This constraint could be enforced at tenant level against trust relations. The role level *SoD* is straightforward. Two roles attached to conflict duties are not supposed to be activated for one user in a session. In the out-sourcing example, a quality assurance (QA) role and a developer role should not be obtained by one user in either tenant, E or OS, at the same time.

Other Considerations. Some other constraints in RBAC [20], such as cardinality and prerequisite roles, are also applicable to tenants. Even if multi-tenancy brings some

complexity to implementation, it is also straightforward to apply these constraints. Moreover, the conflict of interests among tenants can be addressed by Chinese Wall model [23]. For example, two competing tenants cannot be trusted or accessed by tenants of a single issuer.

III. Prototype Implementation and Evaluation

To demonstrate the feasibility of our approach, we implement a prototype to achieve multi-tenant authorization for collaborative cloud services. We also evaluate the performance and scalability of our prototype with CloudStorage [16], an open source cloud storage system, deployed as a cloud service.

1) Implementation Overview: In order to foster fine-grained access control in collaborative cloud environments, we propose Authorization as a Service (AaaS) as a novel service model providing an independent authorization infrastructure in a multi-tenancy manner. This service can be integrated with the existing cloud services to manage and process authorizations for them. AaaS supports multi-tenant access control by applying suitable access control models, such as MT-RBAC.

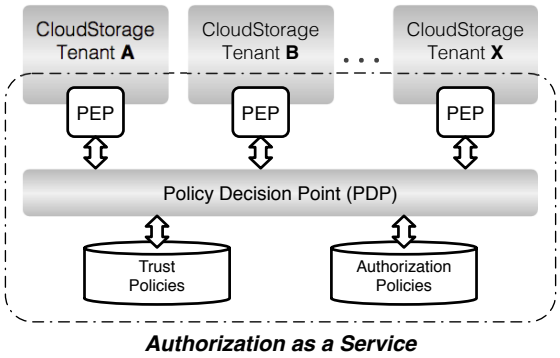


Fig. 4: The prototype architecture

Figure 4 shows the prototype architecture in which the tenants of the cloud storage service use a common authorization service (i.e., AaaS) by associating each with a policy enforcement point (PEP) module. The PEPs parse the requests from end users and generate normalized (XACML format) requests to the centralized policy decision point (PDP) module which refers to MT-RBAC policies stored in the centralized policy repository. The MT-RBAC policy specification is presented in Appendix A. After the decision is made, an XACML format response will be sent back to the requesting PEP to take effect with respect to the requested access. The integrity and authenticity of communication messages should be guaranteed, say via long-lived TLS connections between PEPs and the central PDP. For simplicity, they are not included in the prototype implementation. The prototype can be extended to a cloud authorization service with distributed PDPs.

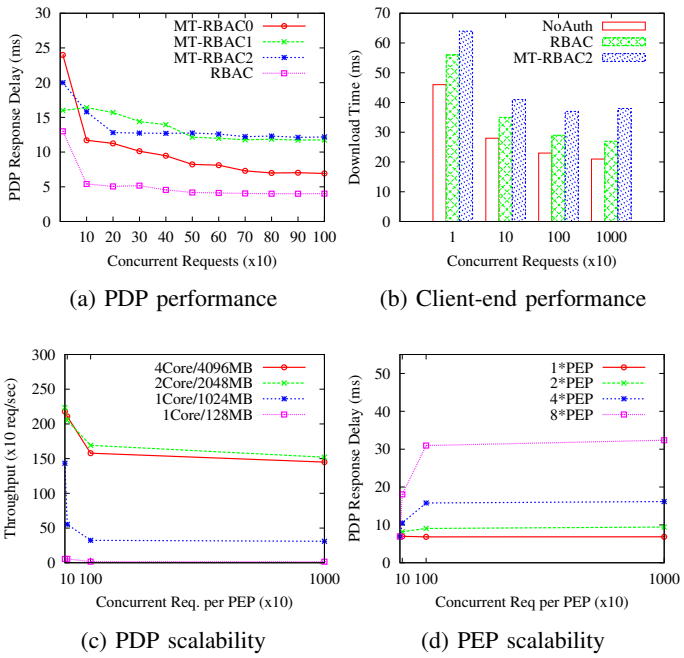
The MT-RBAC policy engine is developed using SUN's XACML library [14]. The prototype is compiled and deployed on a Joyent cloud system. In the performance evaluation, the PDP module runs on a SmartMachine [15] image with SmartOS Version 1.6.3 and 1 GB RAM. The PEP modules and the cloud storage service are deployed on another SmartMachine image with SmartOS Plus 64 Version 3.2.0 with 256 MB RAM. The CPU caps of both images are set to 350 meaning each can use at most 3.5 CPUs. The PDP and PEP modules are created on different physical machines, so that they don't affect each other in the performance evaluation results. All the machines in the prototype are connected through data center networks. End users connect to the cloud storage service through wireless local area network (WLAN) which is a common connection type for cloud service users. Note that, in the following experiments, the policy files are stored on the PDP server and an experiment request set consists of 10 independent sample multi-tenant access requests.

2) Authorization Delay: An MT-RBAC decision making process includes verifying subjects, resources and actions in the request, searching attributes and linking referred policy files. All these operations increase the overhead of access as authorization is involved. Authorization overhead is inevitable in MT-RBAC as well as other authorization models. Figure 5a compares the policy evaluation delay in RBAC and the three MT-RBAC models. Note that in RBAC cross-tenant access requests are not supported. Due to the caching mechanisms of the operating system, as the number of concurrent requests increases, the average policy decision delay decreases dramatically until it reaches a stable state. RBAC has the least delay of 4.01 ms, while MT-RBAC$_0$ has 6.92 ms delay. The evaluation of TP policies contributes to the extra delay of MT-RBAC$_0$, compared to RBAC. Since IPS and DPS evaluations incur the similar I/O operations to TP evaluations, the authorization delay for MT-RBAC$_1$ and MT-RBAC$_2$ are similar. MT-RBAC$_1$ and MT-RBAC$_2$ have the most delay of 11.75 ms and 12.18 ms, respectively. MT-RBAC models introduce acceptable evaluation overheads compared to RBAC.

Figure 5b shows a comparison of delays at the client-end of the CloudStorage service. The delays are observed upon a 1KB file downloading task with or without authorization through RBAC or MT-RBAC$_2$. According to the experiment result, we observe that MT-RBAC$_2$ introduces 15.50 ms (≈ 0.016 second) authorization delay on average which we believe is acceptable for file downloading tasks in cloud storage services.

3) Scalability: Scalability is also a critical criterion to judge the feasibility of a cloud application. Thus, we evaluate our prototype with various cloud images for the PDP module and numbers of engaged PEPs to see whether the system performance improvement is proportional to the increase of the hardware capacity which is represented by image size in the cloud.

We compare the throughput of PDPs with various capabilities in terms of image size. The result is shown in Figure 5c. The speedup of the system is in positive correlation with the increase of CPU cores and memory size. The throughput decreases when the number of concurrent requests increase, until it reaches a stable position. Compare the authorization throughput of the PDP with *1Core/128MB* and the one with *1Core/1024MB*. The approximately ten-time increase in memory size results in constantly ten-time enhancement in the throughput. When the physical resource assigned to the PDP increases to *2Core/2048MB*, the throughput increases around five times. But the throughput has no obvious increase when the hardware becomes *4Core/4096MB* because the amount of requests does not fulfill the utilization of the system with increased capacity. According to the results, we conclude that



(a) PDP performance (b) Client-end performance

(c) PDP scalability (d) PEP scalability

Fig. 5: Performance and scalability evaluation results

the system throughput is proportional to the hardware capacity of the PDP module.

Figure 5d illustrates the authorization delay in a *1Core/1024MB* PDP with different amounts of engaged PEPs. More PEPs means more concurrent requests generated and more connections which consume the capacity of the PDP. When the number of PEPs increase exponentially, the average evaluation delay also increases exponentially. Hence, if all the engaged PEPs send equal amount of requests, the load of PDP can be roughly determined by the number of PEPs. As a PEP is assigned per tenant, the required capacity of the system is proportional to the amount of tenants. The results show that AaaS with MT-RBAC is scalable in the cloud environment.

IV. Related Work

The access control problems in collaboration have been extensively addressed in traditional environments. RBAC [7], [20] enabling fine-grained access control does not encompass the overall context associated with any collaborative activity [24]. Many extensions of RBAC have been proposed to enable multi-domain access control [25]–[28]. These approaches require a centralized authority to define collaborating policies for different domains. However, in clouds, users usually come from different organizations with independent administrating authorities. Therefore, a centralized authority is not only improper but also inefficient.

Another approach is to integrate delegation in RBAC to achieve decentralized authority for collaborations [29]–[32]. Users may delegate their entire or partial permissions to others, which is completely under the control of the users. The delegation approach enables full control for resource providers while introducing additional administration workloads. The administrators of resource providers need to track all of the delegations associated with collaborations. In particular, collaboration relations may dynamically change. The delegation rules may not instantly reflect the current collaboration relations so that delegation rules may violate the authorization goals of the resource providers.

To support collaboration, federated identity and authorization services were proposed in distributed systems. Federated identity [18] enables authenticating strangers by sharing identity information among federated parties who trust each other equally. The equivalent trust relation limits the variety of collaborations. Moreover, maintaining the federation relation becomes costly when it has to cope with the agility feature in clouds. Some tenants are created temporarily and deleted upon completion of their jobs, while the federation relation has to be updated accordingly. Authorization services [33]–[37] were developed to control resource sharing between different *Virtual Organizations* (VOs) in grids leveraging credentials. However, cloud is designed with less heterogeneity than grid for better flexibility and scalability [10] so that the maintenance of credentials becomes a critical problem in cloud settings. Moreover, the centralized facility of clouds provide opportunities for policy-driven authorization services. Therefore, to build collaborations among tenants, such credential-driven approaches are neither appropriate nor necessary.

As a critical part of the evolving cloud technology, authorization mechanisms for multi-tenant collaborations are emerging in both academia and the industry. Multi-tenancy in the cloud features in homogeneity, centralized facility, and agility which the former approaches cannot handle. Similar to AaaS, these approaches [11], [38]–[40] utilize centralized facilities in clouds to develop authorization services with scalable policy management modules and PDPs in a central location. This setting is consistent with MT-RBAC.

Role-Based Trust Models are effective in terms of access control for collaboration in distributed environments. Trust relations, in certain forms, are established among administrative domains for sharing. The effects of different trust models vary in different aspects. We hereby identify some crucial differences among three role base trust models: Role-based Trust-management (RT) framework [13], Multi-Tenancy Authorization System (MTAS) [11], [12], and our approach MT-RBAC in this paper, as shown in Table II.

RT is a family of Role-based Trust-management language using credentials to express trust relations and policies in distributed authorization. In this paper, we only discuss the key features of RT_0 which is the base model in the family. RT_0 provides four types of multi-domain assignments (credentials): simple member, simple inclusion, linking inclusion, and intersection inclusion. The former two are compatible with MTAS and MT-RBAC which do not support the latter two.

First, we compare the models in terms of the trust and authorization assignment authorities as well as the required trust relations for collaborations. In RT the trust assignment and the authorization assignment are coupled in a credential so that both of them are assigned by the resource owner who trusts the requester. Conversely, in MTAS and MT-RBAC the two assignments are decoupled and issued by different parties. The unique feature of MT-RBAC is that the resource owner trusts the requester not only to access the resources but also for its issuer to authorize the access that is to make authorization

TABLE II: Trust Model Comparison. A and B represent two entities, issuers and tenants respectively in RT, MTAS and MT-RBAC. A represents the resource owner and B the requester.

	RT	**MTAS**	**MT-RBAC**
trust relation required	A trust B	B trust A	A trust B
trust assigner	A	B	A
authorization assigner	A	A	B
User Assignment (UA)	$U \rightarrow A.R$	$U \rightarrow A.R$	$B.U \rightarrow B.R \cup A.R$
Permission Assignment (PA)	$A.P \rightarrow A.R$	$A.P \rightarrow A.R \cup B.R$	$B.P \rightarrow B.R$
Role Hierarchy (RH)	$A.R \leq B.R$	$A.R \leq B.R$	$A.R \leq B.R$
require common vocabulary	Yes	No	No
require centralized facility	No	Yes	Yes

assignments for the owner's permissions. Further, the direction of a trust relation is special in MTAS where the resource owner has to be trusted by the resource requester before making appropriate authorization assignments.

Then, the models are examined respectively with the three kinds of authorization assignments: UA, PA, and RH. With UA, properly authenticated users are allowed to be assigned by resource owners to their own roles in RT and MTAS, while in MT-RBAC the resource requester (trustee) can only assign its own users to the roles of itself or the resource owner (truster) who trusts the requester. Cross-domain permission assignments are not allowed in RT or in MT-RBAC, but possible in MTAS. The inheritance of permissions through RH is always from the resource owner to the requester in all three models.

Last but not least, we discuss the necessity of a prerequisite to enable collaborations with role based trust models. Common vocabulary is a term introduced in RT [13] requiring both collaborators of a trust relation to use a mutually understandable definitions of roles. Thus the semantic mismatch issue in decentralized collaboration is mitigated. However, there is no such requirement in MTAS and MT-RBAC since their trust relations allow exposure of the truster's roles to the trustee so that the definition of roles is perceivable to the administrator within a common centralized facility such as an AaaS platform.

Although these role-based trust models foster collaborations in the cloud, a unified and consolidated trust framework is not currently available. We anticipate research in this field will establish foundations for the evolution of cloud computing.

V. Conclusion

In this paper, we propose a family of MT-RBAC models by extending the well-known and widely used RBAC model with the components of *tenants* and *issuers* to address multi-tenant authorization for collaborative cloud services. MT-RBAC aims to enable fine-grained cross-tenant resource access by building tenant-level granularity of *trust relations*. We prototype MT-RBAC using SUN's XACML library to implement an Authorization as a Service (AaaS) in cloud. To demonstrate the viability of the prototype system, we evaluate its performance and scalability in a Joyent cloud. The results show that the AaaS platform with MT-RBAC incur acceptable overhead and is scalable for the cloud storage service.

Acknowledgement

This work is partially supported by grants from the National Science Foundation and AFOSR MURI program.

References

[1] "Gartner outlines five cloud computing trends that will affect cloud strategy through 2015," Gartner Press Release, 2012.

[2] P. Mell and T. Grance, "The NIST definition of cloud computing," Special Publication 800-145, 2011.

[3] J. Judkowitz, "Taking advantage of multi-tenancy to build collaborative clouds," http://communities.intel.com/community/datastack/cloudbuilder/blog/2011/04/29/taking-advantage-of-multi-tenancy-to-build-collaborative-clouds, 2011.

[4] D. Jermyn, "Health care not yet ready to share," https://secure.globeadvisor.com/servlet/ArticleNews/story/gam/20110610/SRCLOUDHEALTH0610ATL, 2011.

[5] A. Kurmus, M. Gupta, R. Pletka, C. Cachin, and R. Haas, "A comparison of secure multi-tenancy architectures for filesystem storage clouds," in *Middleware*, 2011, pp. 471–490.

[6] J. McKenty, "Nebula's implementation of role based access control (RBAC)," http://nebula.nasa.gov/blog/2010/06/03/nebulas-implementation-role-based-access-control-rbac/, 2010.

[7] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 224–274, Aug. 2001.

[8] F. Chong, G. Carraro, and R. Wolter, "Multi-tenant data architecture," http://msdn.microsoft.com/en-us/library/aa479086.aspx, 2006.

[9] R. F. Chong, "Designing a database for multi-tenancy on the cloud," http://www.ibm.com/developerworks/data/library/techarticle/m-1201dbdesigncloud/index.html, 2012.

[10] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *GCE*, 2008, pp. 1–10.

[11] 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, Nov/Dec 2010.

[12] 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.

[13] N. Li, J. C. Mitchell, and W. H. Winsborough, "Design of a role-based trust-management framework," in *IEEE Symp. on Sec. and Priv.*, 2002, pp. 114–130.

[14] "Sun's XACML implementation," http://sunxacml.sourceforge.net/.

[15] "Joyent SmartOS," http://smartos.org/.

[16] "Cloud storage system," http://sourceforge.net/projects/cloudstorage/.

[17] R. Bhatti, E. Bertino, and A. Ghafoor, "X-FEDERATE: A policy engineering framework for federated access management," *IEEE TSE*, vol. 32, pp. 330–346, 2006.

[18] ——, "An integrated approach to federated identity and privilege management in open systems," *CACM*, vol. 50, no. 2, pp. 81–87, 2007.

[19] M. T. Goodrich, R. Tamassia, and D. D. Yao, "Notarized federated ID management and authentication," *J. Comput. Secur.*, vol. 16, no. 4, pp. 399–418, 2008.

[20] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[21] M. Shehab, E. Bertino, and A. Ghafoor, "SERAT: SEcure role mApping technique for decentralized secure interoperability," in *SACMAT*, 2005, pp. 159–167.

[22] "Sarbanes-Oxley Act (SOX)," Public Law 107-204, 2002.

[23] D. Brewer and M. Nash, "The chinese wall security policy," in *IEEE Symp. on Sec. and Priv.*, 1989, pp. 206–214.

[24] W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong, "Access control in collaborative systems," *ACM Comput. Surv.*, vol. 37, no. 1, pp. 29–41, 2005.

[25] E. Cohen, R. K. Thomas, W. Winsborough, and D. Shands, "Models for coalition-based access control (CBAC)," in *SACMAT*, 2002, pp. 97–106.

[26] Q. Li, X. Zhang, M. Xu, and J. Wu, "Towards secure dynamic collaborations with Group-Based RBAC model," *Computers & Security*, vol. 28, no. 5, pp. 260–275, 2009.

[27] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo, "Policy decomposition for collaborative access control," in *SACMAT*, 2008, pp. 103–112.

[28] Z. Zhang, X. Zhang, and R. Sandhu, "ROBAC: Scalable role and organization based access control models," in *CollaborateCom*, 2006, pp. 1–9.

[29] M. Alam, X. Zhang, K. Khan, and G. Ali, "xDAuth: A scalable and lightweight framework for cross domain access control and delegation," in *SACMAT*, 2011, pp. 31–40.

[30] L. Bauer, L. Jia, M. K. Reiter, and D. Swasey, "xDomain: cross-border proofs of access," in *SACMAT*, 2009, pp. 43–52.

[31] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti, "dRBAC: distributed role-based access control for dynamic coalition environments," in *ICDCS*, 2002, pp. 411–420.

[32] X. Zhang, S. Oh, and R. S. Sandhu, "PBDM: a flexible delegation model in RBAC," in *SACMAT*, 2003, pp. 149–157.

[33] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, K. Lőrentey, and F. Spataro, "From gridmap-file to VOMS: managing authorization in a grid environment," *Future Gener. Comput. Syst.*, vol. 21, no. 4, pp. 549–558, 2005.

[34] E. Bertino, P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Ferrari, "Towards supporting fine-grained access control for grid resources," in *FTDCS*, 2004, pp. 59–65.

[35] D. W. Chadwick and A. Otenko, "The PERMIS X.509 role based privilege management infrastructure," in *SACMAT*, 2002, pp. 135–140.

[36] P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Bertino, "Efficient integration of fine-grained access control and resource brokering in grid," *The Journal of Supercomputing*, vol. 49, no. 1, pp. 108–126, 2009.

[37] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, "A community authorization service for group collaboration," in *POLICY*, 2002, pp. 50–59.

[38] H. Takabi and J. B. D. Joshi, "Semantic-based policy management for cloud computing environments," *International Journal of Cloud Computing*, vol. 1, no. 2, pp. 119–144, 01 2012.

[39] A. Almutairi, M. Sarfraz, S. Basalamah, W. Aref, and A. Ghafoor, "A distributed access control architecture for cloud computing," *IEEE Software*, vol. 29, no. 2, pp. 36–44, 2012.

[40] S. Gerges, S. Khattab, H. Hassan, and F. Omara, "Scalable multi-tenant authorization in highly collaborative cloud applications," *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, vol. 2, no. 2, pp. 106–115, 2013.

[41] "Core and hierarchical role based access control (RBAC) profile of XACML v2.0," OASIS Standard, 2005.

APPENDIX

We specify MT-RBAC policies in extensible access control markup language (XACML). According to the normative specification of RBAC policies [41], we keep using the Role PolicySet (RPS) and the Permission PolicySet (PPS), representing UA and PA respectively in MT-RBAC. Additionally, a novel Trust PolicySet (TPS) is added. In order to express MT-RBAC$_1$ and MT-RBAC$_2$ policies, we also introduce $TIPR$ PolicySet (IPS) and $TDPR$ PolicySet (DPS). In this section, we present an MT-RBAC$_2$ policy example and the corresponding authorization process.

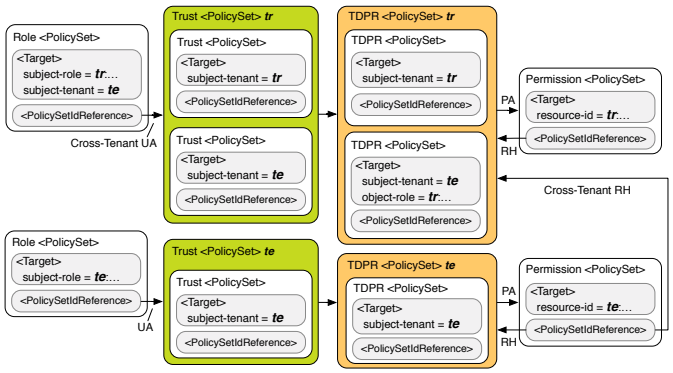


Fig. 6: The MT-RBAC$_2$ policy enables two independent authorization paths: cross-tenant user assignment (*UA*) and cross-tenant role hierarchy (*RH*), where *tr* trusts *te*.

Cross-Tenant UA: starting from the upper RPS in Figure 6, a user in te sends a request to access tr's resource. If the user is already assigned to a role in tr, RPS will forward the request to tr's TPS who checks if $tr \trianglelefteq te$. If the trust relation does not exist, the request will be denied; otherwise, tr's DPS will check if the user's role in tr is in $\mathcal{P}_{TD}(tr, te)$. If the role is public to te, tr's PPS associated with the role will check the resources and actions in the request. If these do not match the policy rules, the request will be forwarded to the DPS again to verify the visibility of the junior roles to te. This process will execute recursively until a match in PPS rules is found or the lowest role visible to te in tr's role hierarchy is reached. If a match is found, the PDP will respond with a permit, otherwise the PDP will check other authorization paths in the policy tree for a match. If finally no match is found, a deny response will be returned to the PEP.

Cross-Tenant RH: starting from the lower RPS in Figure 6, the request from a user of a te's role will first be sent to te's TPS. The te's TPS and the subsequentDPS will forward the user's request because it is an intra-tenant request. Then, the request will arrive to the PPS of the user's role in te. The recursive process of retrieving a proper junior role will take place. Since a cross-tenant RH assignment to a junior role in tr exists, the request will be forwarded to tr's DPS during the recursive process. The steps afterwards is similar as described in the other authorization path.

The authorization paths in the other MT-RBAC models are similar. The IPS in MT-RBAC$_1$ specification enforces $TIPR$ similarly as the DPS in MT-RBAC$_2$ enforcing $TDPR$ in the example above.