Multi-Tenancy Authorization Models for Collaborative Cloud Services

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

***Abstract*—The cloud service model intrinsically caters to multiple tenants, most obviously in public clouds but also in private clouds for large organizations. Currently most cloud service providers (CSPs) isolate user activities and data within a single tenant boundary with no or minimum cross-tenant interaction. It is anticipated that this situation will evolve soon to foster cross-tenant collaboration supported by Authorization as a Service (AaaS). At present there is no widely accepted model for cross-tenant authorization. Recently, Calero et al [12] informally presented a multi-tenancy authorization system (MTAS) which extends the well-known role-based access control (RBAC) model by building trust relations among collaborating tenants. In this paper we formalize this MTAS model and propose extensions for finer-grained cross-tenant trust. We also develop an administration model for MTAS (AMTAS). We demonstrate the utility and practical feasibility of MTAS by means of an example policy specification in XACML. We anticipate researchers will develop additional multi-tenant authorization models before consolidation and unification.**

***Keywords*—Security Models for Cloud Computing; Access Control in Collaboration Environments; Role Based Access Control, Reputation, and Trust; Fundamentals and Frameworks for Security in Collaboration Systems; Privacy Protection for Collaboration Systems**

I. Introduction

As cloud adoption increases, cloud service providers (CSPs) are seeking ways to improve their service capabilities. A natural approach, as the recent trend suggests [31], is to establish collaborative relations among cloud services, especially at the Software as a Service (SaaS) layer [27]. Thereby, the resources of a cloud service are available not only to its original users but also users from other collaborators. Collaboration among cloud services mitigates the data lock-in issue [6] and brings new opportunities for more sophisticated services. However, the mashup of user activities and data across collaborators raises security and privacy issues.

Typically, SaaS CSPs have their services hosted by Platform as a Service (PaaS) clouds in which the SaaS services are treated as tenants and segregated by the multi-tenancy mechanism [27]. Collaborations among tenants require an adaptive access control model. The model has to cope with the different access control mechanisms and policies in different tenants. Moreover, the agility, flexibility and granularity of such a model should also be considered. Clearly, maintenance of sensitive information for each collaborator is crucial.

We identify some characteristics of the cloud environment, along with the corresponding requirements in collaborative access control models, as follows.

- **Centralized Facility.** CSPs typically present an abstraction of their services as a pool of computing resources to their clients. Since the resources are centralized in the cloud, fully decentralized access control models used in traditional distributed environments are not appropriate or suitable.
- **Agility.** A tenant in a cloud may be created for temporary use and deleted afterwards. So access control models in clouds should also be agile and flexible enough to cope with this kind of usage.
- **Homogeneous Architecture.** The services in a cloud are supposed to be equal in quality, as most CSPs build and maintain cloud systems with homogeneous infrastructures while the user configurations are different. Therefore, the access control model in different tenants tend to be similar, especially in SaaS.
- **Out-Sourcing Trust.** Cloud users intrinsically out-source part of their IT infrastructures to CSPs in order to lower the cost so that trust relations between the two parties are already established. Collaborations among tenants also need similar trust relations, which can be developed through their common trust in the CSP.

Currently, CSPs use Single Sign-On (SSO) techniques to achieve authentication and simple authorization in federated cloud environments, but fine-grained authorizations are typically not supported. NASA has integrated role-based access control (RBAC) into Nebula [26], a private cloud system. While traditional RBAC enables fine-grained access control mechanisms in clouds, it lacks the ability to manage collaborations. IBM [15] and Microsoft [14] proposed a resource sharing approach in data-centric clouds using database schema, but this approach is specialized to databases and cannot be directly applied to other types of services. Collaboration models in traditional access control models, such as RT [22] and dRBAC [19], use credentials to securely communicate among collaborators. The management of credentials remains a problem which could be avoided in cloud environments because of the existence of centralized facilities.

To achieve collaborations among cloud services, Calero et al [12] proposed a multi-tenancy authorization system (MTAS) by extending RBAC with a coarse-grained trust relation. The authorization policies and trust assertions are stored in a centralized knowledge base. The authorization decisions are also made in a centralized policy decision point (PDP). Calero et al described an authorization model and a trust model in an informal way, while noting that the trust relation is coarse-grained and open for extensions.

In this paper, we abstract the collaborative access control mechanisms of MTAS in a formal model. Additionally, we propose an administration model for MTAS and build finer-grained enhancements upon the trust model. The administration model formally specifies the administrative functions managing authorization policies and trust assertions with decentralized authority. One enhancement of the trust model introduces truster-centric public role (TCPR) constraints over the trust relation, i.e. a truster only exposes its predefined public roles to its trustees. This approach limits unnecessary disclosure of the trusters' sensitive information in the collaboration processes. Beyond TCPR, we also give an even finer-grained trust model, relation-centric public role (RCPR) by defining public roles with respect to a specific trust relation.

The rest of the paper is organized as the following. Section II presents a use case of multi-tenant collaborations in the cloud and discusses current approaches in context of this example. The formal model of MTAS is presented in Section III along with its administration model and enhancements in the trust model. In Section IV, we describe the policy specification of the MTAS model in XACML as one possible implementation. Section V concludes the paper.

II. Background and Motivation

In order to provide a variety of services, collaborations are increasingly common in IT systems, especially in distributed systems. Yet, collaborations among services are not fully supported in today's cloud environment. In part, due to this lack, data lock-in issues are rated as second of the top ten issues for cloud computing adoption [6]. User data is usually contained within one service and not easily used in others. This results in inconvenience and waste of resources. For example, a user may want to open one of their own files stored in Dropbox directly on the cloud, but Dropbox does not support this function. To achieve this result, a common approach is that the user downloads the file to their local machine and uploads it to another cloud service. In this way, the barrier between the two cloud services is mitigated by the intermediate local machine with extra communications, operations and storage space. Directly building collaborations across these current barriers may be a more effective solution.

A. Case Study

Out-sourcing is the essence of cloud computing. The trust relation between cloud users and CSPs is very similar to the familiar trust relation between organizations and their contracted out-sourcing companies. We use a typical out-sourcing case, as described in the following, to explain the models.

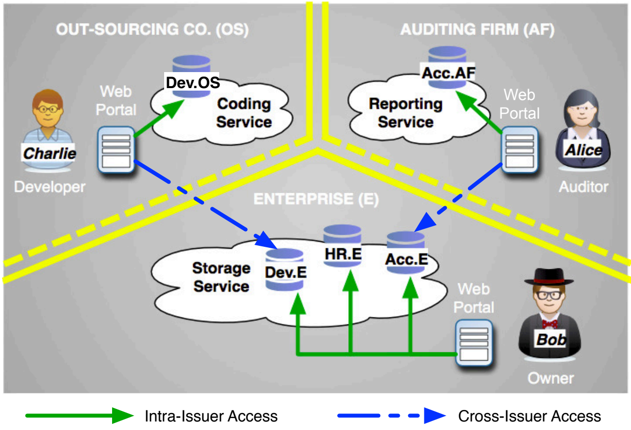

Figure 1. An out-sourcing case of multi-tenant accesses.

Figure 1 presents the out-sourcing case. Enterprise (E), Out-Sourcing Company (OS) and Auditing Firm (AF) are three independent organizations using cloud storage service, coding service and reporting service respectively. The yellow lines represent the cloud service boundaries. Similar to the pavement markings, the double solid line means "do not pass" and the double line with one side solid and the other side broken means "one way pass only" from the broken side. Let "." denote the affiliation relation between a tenant and an organization, e.g., $Dev.E$ represents a development tenant on the cloud storage service of E. As some of E's application development is out-sourced to OS, the developer $Charlie$ from OS is authorized to access the source code stored in $Dev.E$. In the meanwhile, E has a contracted AF to execute external auditing of E's financial and application development projects on a regular basis, so that the auditor $Alice$ from AF is allowed to have read-only accesses to both $Acc.E$ and $Dev.E$. The human resource information of E is stored in $HR.E$ which is not accessible externally.

B. Current Approaches

Access control problems in collaborative environments have been extensively addressed in the research community. Many extensions of RBAC [17], [29] have been proposed to enable multi-domain access control [16], [23], [24], [33]. In these approaches, the presence of a centralized authority is required. It acts as an administrator to manage collaborative policies among domains. However, in clouds, typical issuers come from different organizations with independent administrative authorities. Therefore, centralized authority may not be suitable for the cloud.

Another line of work seeks to integrate delegation in RBAC, in order to obtain decentralized authority in collaborations [4], [7], [8], [19], [32]. Users may delegate their entire or partial roles to others, entirely at their discretion within constraints established by the security architects. This fragments the authorization on basis of individual user decisions which may lead to lack of agility as authorization goals change.

To support collaboration, federated identity and authorization services were proposed in distributed environments. Federated identity [10] enables authenticating strangers by sharing identity information among federated parties who trust each other equally. Moreover, the establishment and maintenance of federations has proved to be costly and far from agile. Authorization services [5], [9], [13], [25], [28] were developed to control resource sharing between different *Virtual Organizations* (VOs) in grids utilizing asymmetric-key based credentials. However, the cloud is designed with centralized facility and less heterogeneity than the grid for better flexibility and scalability [18]. Therefore, such costly and inefficient credential-driven approaches are not necessary to build collaborations in clouds.

By introducing trust management into access control mechanisms [3], [20]–[22], decentralized authority is achieved. However, these approaches needs to build extra facilities or changing the existing administrative models, in order to cope with the semantic mismatch issue.

C. Authorizaiton as a Service (AaaS)

In the cloud environment, multi-tenant architecture brings new challenges to collaborative authorization. The homogeneous architecture and centralized facility characteristics of the cloud differentiate it from traditional distributed environments. In order to address access control problems in the cloud, we build upon the concept of Authorization as a Service (AaaS). Similar to other service models, AaaS is an independent framework providing authorization service to its clients in a multi-tenant manner while the service itself is managing access control for the tenants. The authorization policies of the tenants are stored separately in a centralized facility where a policy decision point (PDP) is able to collect necessary policies and attributes it needs to make appropriate authorization decisions. In this framework, a general access control model is required.

III. Formalized Models

In this section we formalize the multi-tenancy authorization system informally described in [12]. We call the resulting model as the MTAS model for ease of reference and continuity. We also introduce an administration model for MTAS (called the AMTAS model). Further, we propose two feasible enhancements to the trust model of MTAS.

A. Overview

The MTAS model is abstracted from the MTAS system, as shown in Figure 2. There are four entity components: *issuers* (I), *users* (U), *permissions* (P) and *roles* (R). In addition to classic RBAC$_2$, the role hierarchy model [17], the *issuer* component is introduced to express authorization in multi-tenant environments, while other components need to be modified accordingly. In particular the traditional RBAC entities of *permissions* and *roles* have *issuer* attributes so that they can be identified uniquely in a multi-tenant cloud environment. This is depicted by the RO and PO relations in Figure 2. RO and PO are many-to-one relations from R and P respectively to I.

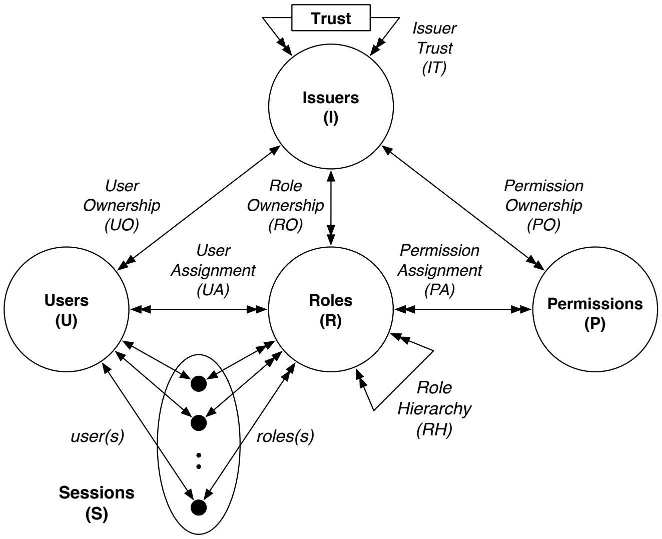

Figure 2. An abstracted model of the MTAS system.

ISSUERS. An *issuer* represents an organization or an individual who uses the cloud services. It is a client of the CSPs'. An *issuer* may use multiple cloud services and vice versa. A service creates an interface (tenant) for each *issuer* so that the data and action of the *issuer* are isolated from each other. For example, in the out-sourcing case, E is an issuer who owns three tenants: $Dev.E$, $Acc.E$ and $HR.E$. The tenants are operated separately.

USERS. A *user* is an identifier for an individual (or a process). It is authenticated as a federated ID [10] which is universally unique for all the issuers in the community. Every user has an owning issuer who provides the identity and authentication of the user. The identity is also usable by other issuers.

PERMISSIONS. A *permission* is a specification of a privilege to an object on a tenant, which is specified as a service interface. A permission is denoted in a 3-tuple *(privilege, tenant, object)*. For example, *(read, Dev.E, /root/)* represents a *permission* of reading the "/root/" path on $Dev.E$. Because the tenant attribute of a permission belongs to only one *issuer*, every *permission* is associated with a single *issuer* while one issuer may have multiple permissions.

ROLES. A *role* is a job function (role name) with an issuer. A role is denoted as *role(issuer, roleName)*, e.g. $role(E, dev)$ represents a developer role in issuer E. A *role* belongs to a single *issuer* while an *issuer* may own multiple *roles*.

SESSIONS.[a] 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*. In a multi-tenant cloud environment,

[a] The session component was not discussed in [12], but we feel it indispensable in a complete formal model which builds on RBAC, so it is included and some session related components are added in the formalization, as described in Definition 2.

note that the user and the active roles in a *session* might not all be from the same issuer.

Crucially, an additional issuer trust relation ($IT \subseteq I \times I$, also written as "$\lesssim$") establishes issuer to issuer trust as will be described and formalized in detail later in this section. For $\forall i_r, i_e, i_f \in I$, IT relation is reflexive

$$i_r \lesssim i_r \tag{1}$$

but not transitive

$$i_r \lesssim i_e \wedge i_e \lesssim i_f \nRightarrow i_r \lesssim i_f \tag{2}$$

and it is neither symmetric

$$i_r \lesssim i_e \nRightarrow i_e \lesssim i_r \tag{3}$$

nor anti-symmetric

$$i_r \lesssim i_e \wedge i_e \lesssim i_r \nRightarrow i_r = i_e. \tag{4}$$

For $\forall(i_r \lesssim i_e)$, we call i_r the truster and i_e the trustee. In MTAS model, trust is always established by the truster allowing the trustee to view and use its own authorization statements. Therefore, the trustee can grant one of the truster's roles, say r_2, a trustee's permission, say p_1. This role to permission assignment enables all users in r_2 to inherit p_1. Further the trustee can make one of the trustee's roles, say r_1 to be junior to one of the truster's roles, say r_2. The effect of this role to role assignment is to make all users in r_2 members of r_1 so that the permissions of r_1 in the trustee are also inherited by the users of r_2 in the truster. The definition of MTAS trust model is given below.

Definition 1: Let A and B denote two issuers. By establishing a issuer trust relation (IT) with B ($A \lesssim B$), A exposes its entire role hierarchy and the role members to B so that B is able to make the two following assignments:

1) assigning B's permissions to A's roles; and
2) assigning B's roles as junior roles to A's roles.

For example, in the out-sourcing case as described in Section II-A, Bob, representing the resource owner E, should allow certain developers in OS to access the source code files stored in $Dev.E$ for them conduct the out-sourcing job. Assume the proper permission in E for the out-sourcing job, *(edit, Dev.E, /src/)* is associated to the role $role(E, dev)$. In order to achieve this cross-issuer access, with the presence of $OS \lesssim E$ relation, Bob can assign $role(E, dev)$ to be a junior role of a proper developer role in OS, say $role(OS, dev)$. In this way, the users associated to $role(OS, dev)$ are able to edit the files under the */src/* directory in $Dev.E$.

The trust model solves the two key problems in collaborative role-based access control: decentralized authority and semantic mismatch. Since the collaborators are independent self-managing services, the service issuers (decentralized authorities) desire to remain control of their resources including data and authorization settings. But in most collaborations, some level of resource sharing is inevitable and that is why we need a trust model to keep the resource sharing process secure. By establishing a trust relation described in Definition 1, the truster exposes its authorization settings to the trustee while the trustee assigns permissions of its data to the truster. In this way, both sides contribute to cross-issuer assignments and the accesses are under mutual control.

The semantic mismatch issue refers to the fact that the definitions of roles vary in different domains so that no proper assignment could be made by a single authority without additional communication with each other. In the trust model of MTAS, this issue is mitigated, because the authorization settings, i.e. the role hierarchy and the role members, of the truster are exposed to the trustee upon the creation of the issuer trust relation. Consider the out-sourcing case. With the presence of $OS \lesssim E$, E's administrator may search the members of OS's roles and decide which role is appropriate to assign the permission to.

B. MTAS Model

The formal definition of MTAS model is as follows.

Definition 2: The MTAS authorization model has the following components:

- U, R, P, I and S (users, roles, permissions, issuers and sessions respectively);
- $UO \subseteq U \times I$, a many-to-one relation mapping each user to its owning issuer;
- $RO \subseteq R \times I$, a many-to-one relation mapping each role to its owning issuer; correspondingly, $roleOwner(r : R) \rightarrow I$, a derived function mapping a role to its issuer where $roleOwner(r) \in \{i \in I | (r, i) \in RO\}$;
- $PO \subseteq P \times I$, a many-to-one relation mapping each permission to its owning issuer; correspondingly, $permOwner(p : P) \rightarrow I$, a derived function mapping a permission to its issuer where $permOwner(r) \in \{i \in I | (p, i) \in PO\}$;
- $IT \subseteq I \times I$, a reflexive relation on I called issuer trust relation, also written as $\lesssim$;
- $canUse(r : R) \rightarrow 2^I$, a derived function mapping a role to a set of issuers who can use the particular role. Formally, $canUse(r) = \{i \in I | roleOwner(r) \lesssim i\}$;
- $UA \subseteq U \times R$, a many-to-many user-to-role assignment relation;
- $PA \subseteq P \times R$, a many-to-many permission-to-role assignment relation requiring $(p, r) \in PA$ *only if* $permOwner(p) \in canUse(r)$;
- $RH \subseteq R \times R$ is a partial order on R called role hierarchy or role dominance relation, also written as $\geq$, requiring $r \geq r_1$, only if $roleOwner(r_1) \in canUse(r)$;
- $user(s : S) \rightarrow U$, 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 within s, and s has the permissions $\bigcup_{r \in roles(s)} \{p | (p, r) \in PA]\}$.

Note that since we are formalizing an extension of pure RBAC model [17], the user permission assignment described in [12] is ignored in the formalization.

Role activation mechanisms determine the executable permissions inherited by 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. Theoretically the former scenario is transformable to the latter by recursively executing explicit activation for the junior roles. The choice between the two approaches is left as an implementation issue in the NIST RBAC model [17]. In the RBAC96 model implicit activation is specified [29]. In MTAS we choose to specify explicit activation in the $roles(s)$ component. In a session, only the permissions of the explicitly activated roles are executable to the user.

Since a users identity is available for all the issuers, UA assignments have no requirements about where the users come from. Thus, the UA assignments are always issued by the role owner. Administration of the MTAS model is discussed in Section III-C.

The trust model is embedded in the $canUse$ function which takes effect in PA and RH assignments in the MTAS model. As the name suggests, the $canUse(r)$ function returns the issuers who can use r to make authorization assignments. The returned issuers are the trustees who are trusted by r's owner, say i. In order to issue PA, permission owner has to be i itself or one of the trustees of i. Therefore, r is only assigned to permissions of i or its trustees. Similar conditions require that only the roles of i or its trustees can be assigned as junior roles of r in RH. PA and RH assignments enable collaborations among issuers.

Based on the formalization of MTAS model, we also develop a formal administrative model and finer-grained trust models, as presented in the following sections.

C. Administrative MTAS (AMTAS) Model

The administration model, AMTAS is tightly coupled with the MTAS model, since the main problem of access control models in distributed environments is how to manage the decentralized administrative authority. In other words, the administrative model regulates who are eligible to issue what kind of assignments. Hence, a desirable administrative model should maintain balanced management workload and proper control for both sides.

Definition 3: The Administrative MTAS (AMTAS) model requires that

- the resource requester A is responsible for managing the trust relation of $A \lesssim B$; and
- the resource owner B is responsible for managing the assignments (i.e. PA and RH) to A's requesting roles, according to MTAS in Definition 2.

As described in Definition 3, in AMTAS the resource requester maintains full control of the trust relation which is fundamental for cross-tenant accesses through MTAS. The resource owner keeps the ultimate authority of its resources and issues assignments based on properly created and maintained trust relations. Both the trust relations and the assignments are crucial in cross-tenant authorizations, because if either is revoked or altered, the corresponding collaborative accesses will be denied.

TABLE I
ADMINISTRATION FUNCTIONS OF AMTAS FOR ISSUER i

Function	Condition	Update
$assignUser$ (i, r, u)	$i = roleOwner(r) \wedge$ $u \in U$	$UA' =$ $UA \cup \{u \rightarrow r\}$
$revokeUser$ (i, r, u)	$i = roleOwner(r) \wedge$ $u \in U \wedge$ $u \rightarrow r \in UA$	$UA' =$ $UA \setminus \{u \rightarrow r\}$
$assignPerm$ (i, r, p)	$i = permOwner(p) \wedge$ $i \in canUse(r)$	$PA' =$ $PA \cup \{p \rightarrow r\}$
$revokePerm$ (i, r, p)	$i = permOwner(p) \wedge$ $i \in canUse(r) \wedge$ $p \rightarrow r \in PA$	$PA' =$ $PA \setminus \{p \rightarrow r\}$
$assignRH$ (i, r_1, r)	$i = roleOwner(r) \wedge$ $i \in canUse(r_1) \wedge$ $\neg(r_1 \gg r) \wedge$ $\neg(r \geq r_1)$ [a]	$\geq' = \geq \cup \{r_2, r_3 :$ $R \mid r_2 \geq r_1 \wedge r \geq$ $r_3 \wedge$ $roleOwner(r_3) \in$ $canUse(r_2) \bullet$ $r_2 \rightarrow r_3\}$
$revokeRH$ (i, r_1, r)	$i = roleOwner(r) \wedge$ $i \in canUse(r_1) \wedge$ $r_1 \gg r$ [b]	$\geq' = (\gg \setminus \{r_1 \rightarrow$ $r\})^*$ [c]
$assignTrust$ (i, i_1)	$i_1 \in I$	$\lesssim' = \lesssim \cup \{i \rightarrow i_1\}$
$revokeTrust$ (i, i_1)	$i_1 \in I \wedge$ $i \lesssim i_1 \wedge i \neq i_1$	$\lesssim' = \lesssim \setminus \{i \rightarrow$ $i_1\}$ [d]

a. This condition avoids cycle creation in the role hierarchy.
b. It requires r_1 to be an immediate ascendant of r.
c. Implied relations are preserved after revocation.
d. By revoking the trust relation, the canUse() function of i's roles automatically updates accordingly, same as PA and RH.

Table I formally specifies the exact administration functions of AMTAS along with the corresponding conditions and updates to MTAS authorizations.

D. Enhanced Trust Models

The trust model discussed in Definition 1 enables collaborative access control among issuers. However, the unnecessary exposure of the truster's authorization settings raises privacy issues. Therefore, we propose two natural enhancements to the trust model.

1) Truster-Centric Public Role (TCPR): As the name suggests, TCPR introduces the public role constraint for trusters. The public roles are included in a predefined subset of a truster's roles exposed to all of the trustees. It is formally defined as follows.

Definition 4: The truster-centric public role (TCPR) model inherits all the components from MTAS in Definition 2, while the following modifications are applied:

- $\mathcal{P}_T(i : I) \rightarrow 2^R$, a function mapping an issuer to a set of its public roles which are the only roles that i expose to its trustees; and
- $canUse(r : R) \rightarrow 2^T$ is modified to $canUse(r) = \{i\} \cup \{i_1 \in i | i \lesssim i_1 \wedge r \in \mathcal{P}_T(i)\}$, where $i = roleOwner(r)$.

By introducing $\mathcal{P}_T(i)$, the exposure surface of the i's roles in TCPR is much smaller than that in MTAS trust model.

Accordingly, only if $r \in \mathcal{P}_T(i)$, then r can be used by i's trustees. Otherwise, it can only be used internally by i .

Since the public roles in TCPR are defined in terms of the truster i, if $\mathcal{P}_T(i)$ is modified, then all the trust relations with the common truster are influenced. Hence, in practice $\mathcal{P}_T(i)$ tends to contain more public roles than necessary to make sure the availability of all the collaborations that i is using. Therefore, we give a more fine-grained enhancement to the trust model.

2) Relation-Centric Public Role (RCPR): In contrast with TCPR, RCPR enforces the public role constraints for trust relations instead of trusters. The public roles are included in a predefined subset of the truster's roles exposed to the trustee in a specific trust relation. The formal definition follows.

Definition 5: The relation-centric public role (RCPR) model inherits all the components from MTAS in Definition 2, while the following modifications are applied:

- $\mathcal{P}_R(t : IT) \to 2^R$, a function mapping a issuer trust relation to a set of the truster's public roles; and
- $canUse(r : R) \to 2^T$ is modified to $canUse(r) = \{i\} \cup \{i_1 \in I | i \lesssim i_1 \wedge r \in \mathcal{P}_R(i \lesssim i_1)\}$, where $i = roleOwner(r)$.

In RCPR, the public roles of the truster are defined per trust relation so that the role exposure of the truster is accurately expressed and enforced. With this fine-grained constraint, MTAS systems may achieve minimum exposure of the truster's roles in collaborations.

E. Discussion

We now identify several issues introduced by extending RBAC to the multi-tenant environment and discuss potential constraints to mitigate these issues.

1) Cyclic Role Hierarchy: The cyclic role hierarchy is a well known issue in inter-domain access control [30]. A "role cycle" may be formed across tenants in MTAS systems without proper constraints. This may lead to implicit role upgrades in the role hierarchy or other inconsistencies. In order to prevent the formation of role cycles, constraints should be enforced over assignments or sessions. The former is achieved by checking role cycles whenever a cross-tenant RH assignment is issued. Even if there are role-cycles in assignments, the latter prohibits all the roles in a cyclic hierarchy from being activated in the same session. Note that AMTAS includes these provisions.

2) Separation of Duties: During collaborations with MTAS, we have two levels of separation of duties (SoD), issuer level and role level. For issuer level SoD, one collaborating issuer cannot execute two conflict responsibilities. For instance, SOX [1] compliant companies are not suppose to hire the same third-party as both consultant and auditor. This constraint could be enforced over trust relations. The role level SoD is straightforward. Two roles attached to conflict duties are not suppose to be activated for one user in a session. In the outsourcing example, a QA role and a developer role in either issuer, E or OS, should not be obtained by a single user in a same session.

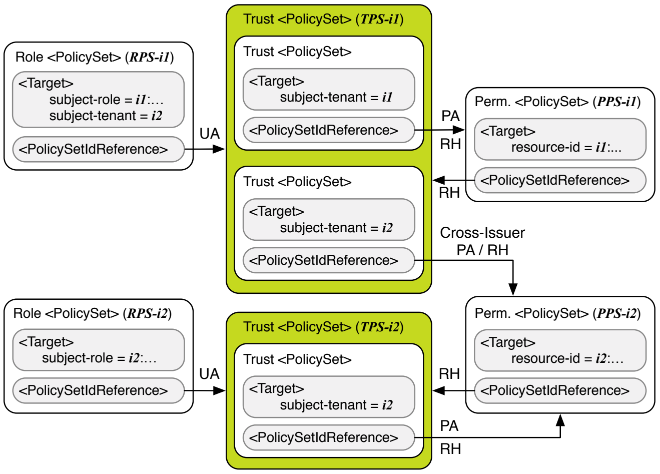

Figure 3. Two authorization paths for $i1$'s user to inherit $i2$'s permission through cross-issuer PA and cross-issuer RH, where $i1 \lesssim i2$.

3) Chinese Wall: The conflict of interests among issuers also needs to be managed. For example, two competing issuers should not be trusted by a single issuer so that the security and privacy of the trustee issuer's sensitive information are protected against the competitors. This situation is already abstracted and addressed by the Chinese Wall model [11] which can be integrated in the centralized AaaS platform to avoid conflict of interests. Essentially, the issuers are grouped into "conflict of interest classes" and by mandatory ruling all issuers are allowed to trust at most one issuer belonging to each such conflict of interest class. In this way, no cross-issuer access will be assigned or permitted by the other conflict of interest issuers.

IV. Policy Specification

In order to demonstrate the feasibility of the MTAS model, we give the policy specification here in the extensible access control markup language (XACML). The normative specification of RBAC policies with XACML2.0 language has been proposed by OASIS XACML TC [2]. Its Role PolicySet (RPS) and Permission PolicySet (PPS), representing UA and PA respectively, are also inherited into the MTAS policy specification. Additionally, a novel Trust PolicySet (TPS) is proposed to express the trust relation. It mediates the two RPS and PPS from multiple issuers.

Figure 3 shows an example of MTAS policy structure. There are two authorization paths for cross-issuer accesses. One of them is cross-issuer permission assignment which starts from the RPS-$i1$. For instance, a user from one of $i1$'s roles sends a request to access $i2$'s resources. RPS will check the role membership of the user. If the user is a member of the claimed role in $i1$, then the request will be forwarded to $TPS-i1$ who checks if $i1 \lesssim i2$. If the trust relation exists, the request will be transferred to DPS-$i2$ which will check if the requested permission is granted to the role. If the answer is true, then the PDP will respond with permit, otherwise the PDP will check other paths of nodes in the policy tree for a match. If finally no match is found, a deny response will be returned.

The other authorization path, as known as cross-issuer role hierarchy assignment, also starts from $RPS\text{-}i1$ in Figure 3. If the user has the membership of a role in $i1$ which is senior to a role in $i2$, then the request will first be forwarded to $TPS\text{-}i1$ which will forward request to $PPS\text{-}i1$. Then $PPS\text{-}i1$ will interact with $TPS\text{-}i1$ recursively to traverse the junior roles until a leaf role is reached or the request is redirected to $PPS\text{-}i2$ by $TPS\text{-}i1$. Then $PPS\text{-}i2$ will run through similar process with $TPS\text{-}i1$ until the requested permission is found and a permit is returned; otherwise the PDP will search all other trustee for the requested permission. If the permission is not found eventually, a deny will be responded.

In summary, through either of the authorization paths, a cross-issuer access control is obtained. The policy specification could be directly used in MTAS implementations.

V. Conclusion and Future Work

To support collaboration between cloud services, we formalize a MTAS model based on a informally specified multi-tenancy authorization system [12] which extends the RBAC model by building trust relations among collaborating services. Further, we give the administration model (AMTAS) and enhancements (TCPR and RMTASCPR) for the trust model in MTAS. In order to demonstrate that MTAS is a viable collaborative AaaS model, we give an example of policy specification in XACML.

Currently, our research team is working towards various collaborative access control models within AaaS framework. Since we use trust relations to achieve collaborations among cloud services, further research in feasible trust models and a potential trust framework is anticipated to emerge from this line of research.

References

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

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

[3] A. K. Adams, A. J. Lee, and D. Mossé. Receipt-mode trust negotiation: efficient authorization through outsourced interactions. In *ASIACCS*, pages 430–434, 2011.

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

[5] 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.*, 21(4):549–558, 2005.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical report, EECS Department, University of California, Berkeley, Feb 2009.

[7] E. Barka and R. Sandhu. Framework for role-based delegation models. In *ACSAC*, pages 168 –176, 12 2000.

[8] L. Bauer, L. Jia, M. K. Reiter, and D. Swasey. xDomain: cross-border proofs of access. In *SACMAT*, pages 43–52, 2009.

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

[10] R. Bhatti, E. Bertino, and A. Ghafoor. An integrated approach to federated identity and privilege management in open systems. *CACM*, 50(2):81–87, 2007.

[11] D. Brewer and M. Nash. The chinese wall security policy. In *IEEE Symp. on Sec. and Priv.*, pages 206–214, 1989.

[12] J. M. A. Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray. Toward a multi-tenancy authorization system for cloud services. *Secur. Priv., IEEE*, Nov/Dec 2010:48–55, 2010.

[13] D. W. Chadwick and A. Otenko. The PERMIS X.509 role based privilege management infrastructure. In *SACMAT*, pages 135–140, 2002.

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

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

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

[17] 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.*, 4(3):224–274, Aug. 2001.

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

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

[20] J. Jin and G.-J. Ahn. Role-based access management for ad-hoc collaborative sharing. In *SACMAT*, pages 200–209, 2006.

[21] J. Jin, G.-J. Ahn, M. Shehab, and H. Hu. Towards trust-aware access management for ad-hoc collaborations. In *CollaborateCom*, pages 41–48, 2007.

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

[23] Q. Li, X. Zhang, M. Xu, and J. Wu. Towards secure dynamic collaborations with Group-Based RBAC model. *Computers & Security*, 28(5):260–275, 2009.

[24] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. Policy decomposition for collaborative access control. In *SACMAT*, pages 103–112, 2008.

[25] 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*, 49(1):108–126, 2009.

[26] 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/.

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

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

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

[30] M. Shehab, E. Bertino, and A. Ghafoor. SERAT: SEcure role mApping technique for decentralized secure interoperability. In *SACMAT*, pages 159–167, 2005.

[31] J. Singhal, S. Chandrasekhar, T. Ge, R. Sandhu, R. Krishnan, G.-J. Ahn, and E. Bertino. Collaboration in multicloud computing environments: Framework and security issues. *IEEE Computer*, 46:76–84, 2 2013.

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

[33] Z. Zhang, X. Zhang, and R. Sandhu. ROBAC: Scalable role and organization based access control models. In *CollaborateCom*, pages 1–9, 2006.