CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2015; **27**:2851–2868
Published online 28 November 2014 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3446

SPECIAL ISSUE PAPER

Multi-tenancy authorization models for collaborative cloud services

Bo Tang[1,2], Ravi Sandhu[1,2] and Qi Li[3,1,*,†]

[1]*Institute for Cyber Security, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, USA*
[2]*Department of Computer Science, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, USA*
[3]*Graduate School at Shenzhen, Tsinghua University, Shenzhen, 518055, China*

SUMMARY

The cloud service model intrinsically caters to multiple tenants, most obviously not only in public clouds but also in private clouds for large organizations. Currently, most cloud service providers 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. At present, there is no widely accepted model for cross-tenant authorization. Recently, Calero *et al.* informally presented a multi-tenancy authorization system (MTAS), which extends the well-known role-based access control 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. We demonstrate the utility and practical feasibility of MTAS by means of an example policy specification in extensible access control markup language. To further test the metrics of the model, we develop a prototype system and conduct experiments on it. The result shows that the prototype has 12-ms policy decision overhead on average and is scalable. We anticipate that researchers will develop additional multi-tenant authorization models before eventual consolidation and convergence to standard industry practice. Copyright © 2014 John Wiley & Sons, Ltd.

Received 10 March 2014; Revised 20 October 2014; Accepted 5 November 2014

KEY WORDS: 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

1. 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 [1], is to establish collaborative relations among cloud services, especially at the Software as a Service (SaaS) layer [2]. Thereby, the resources of a cloud service are available not only to its original users but also to users from other collaborators. Collaboration among cloud services mitigates the data lock-in issue [3] 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 clouds in which the SaaS services are treated as tenants and segregated by the multi-tenancy mechanism [2]. 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

*Correspondence to: Qi Li, Graduate School at Shenzhen, Tsinghua University, Shenzhen 518055, China.
†E-mail: qi.li@sz.tsinghua.edu.cn

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. Because 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 on demand.
- 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 tends 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 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 [4], a private cloud system. Although traditional RBAC enables fine-grained access control mechanisms in clouds, it lacks the ability to manage collaborations. IBM [5] and Microsoft [6] 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 [7] and dRBAC [8], 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.* [9] 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 (AMTAS) 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, that is, 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. To further investigate the feasibility of MTAS, we develop MTAS policy specification in extensible access control markup language (XACML) and a prototype system in a private cloud environment.

The rest of the paper is organized as follows. Section 2 presents the 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 3 along with its administration model and enhancements in the trust model. In Section 4, we use an example to walk through our MTAS policy specification in XACML. Our prototype implementation and benchmarking experiments are described in Section 5. Section 7 concludes the paper.

2. 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, because of this lack, data lock-in issues are rated as second of the top 10 issues for cloud computing adoption [3]. User data are 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.

2.1. *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.

In the out-sourcing example as shown in Figure 1, Enterprise (E), Out-Sourcing (OS) Company, 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, for example, $Dev.E$ represents the 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.

2.2. *Current approaches*

Access control problems in collaborative environments have been extensively addressed in the research community. Many extensions of RBAC [10, 11] have been proposed to enable multi-domain access control [12–15]. In these approaches, the presence of a centralized authority is required. It acts as an administrator to manage collaborative policies among domains. However,

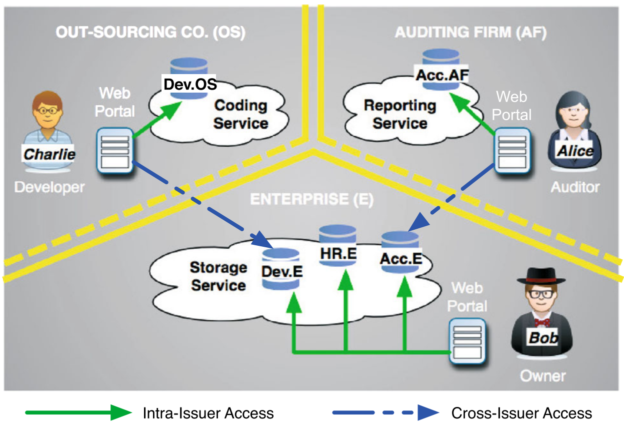


Figure 1. An out-sourcing case of multi-tenant accesses.

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 [8, 16–19]. 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 the 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 [20] enables authenticating strangers by sharing identity information among federated parties who trust each other equally.

Moreover, the establishment and maintenance of federations have proved to be costly and far from agile. Authorization services [21–25] were developed to control resource sharing between different virtual organizations 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 [26]. Therefore, such costly and inefficient credential-driven approaches are not necessary to build collaborations in clouds.

By introducing trust management into access control mechanisms [7, 27–29], decentralized authority is achieved. However, these approaches need to build extra facilities or changing the existing administrative models in order to cope with the semantic mismatch issue.

2.3. *Authorization 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 AaaS. Similar to other service models, AaaS is an independent framework providing authorization service to its clients in a multi-tenant manner, whereas 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 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.

2.4. *Scope and assumptions*

The following assumptions define the scope of this paper.

One cloud service. For simplicity, our work is aimed at addressing access control problems for multiple tenants on a single cloud service. Thus, we intentionally rule out the issue of heterogeneous architecture and incompatible APIs. Although we believe that the proposed models are extensible beyond a single cloud, multi-cloud problems are not considered in this paper.

Authenticated users. In the cloud environment, all the access requesting users are assumed to have been properly authenticated. Particularly, we assume that each user possesses one or more credentials from its correlated identity provider who authenticates the user and includes user information in the credentials. Then, the credentials can be handed to cloud services to request accesses.

Specific trust relation. The tenant trust relations are specifically between two tenants. Other trust relations for more than two tenants, such as federation [30, 31] relations, are not considered in this paper. Each trust relation is unidirectional (like follow in Twitter) as opposed to bidirectional (like friend in Facebook). Also, we assume that each trust relation is established unilaterally by the trustor and remains under exclusive control of the trustor. Specifically, the trustor and only the trustor can create and revoke a trust relation. In other words, the trustee does not need to agree on the establishment or removal of a trust relation.

For simplicity, the trust relation in this paper is a Boolean variable, meaning, a tenant can either trust or not trust another tenant. Other trust relations may be able to express trust levels, and the values can be automatically generated through negotiation or predefined policies. But they are considered out of the scope of this paper.

3. MTAS MODELS

In this section, we formalize the MTAS informally described in [9]. We call the resulting model as the MTAS model for ease of reference and continuity. We also develop an AMTAS model. Further, we propose two feasible enhancements to the trust model of MTAS.

3.1. Formalization

Toward a general model of multi-tenant RBAC in the cloud, we start from abstracting the MTAS system [9] into a formalized model, 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 (RH) model [10], the *issuer* component is introduced to express authorization in multi-tenant environments, whereas 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 role ownership (RO) and permission ownership (PO) relations in Figure 2. RO and PO are many-to-one relations from R and P, respectively, to I.

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 identity for an individual (or a process). It is authenticated as a federated ID [20], which is universally unique for all the issuers in the community. Every user has an owner attribute indicating the 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 three-tuple *(privilege, tenant, and 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*, whereas 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)*, for example, $role(E, dev)$ represents a developer role in issuer E. A *role* belongs to a single *issuer*, whereas an *issuer* may own multiple *roles*.

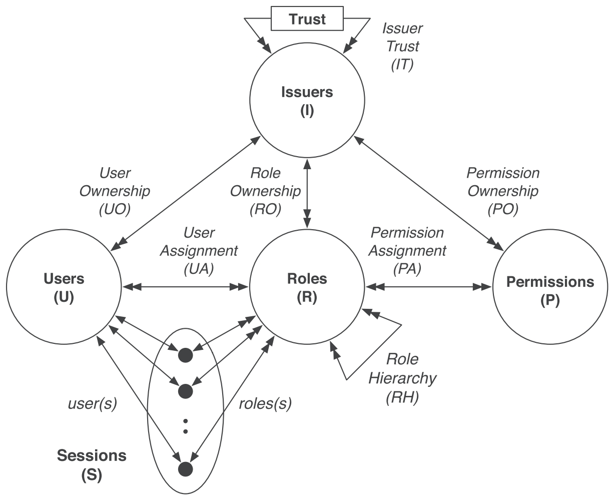


Figure 2. An abstracted model of the MTAS system.

SESSIONS.[‡] A *session* is an instance of access created by a *user*. The owner attribute of the *user* is inherited to the *session*. A session, in its lifespan, is regarded as the subject of the access. A subset of *roles* that the *user* is assigned to be activated in a *session*. In a multi-tenant cloud environment, note that the user and the active roles in a *session* might not all be from the same issuer.

Crucially, an additional issuer trust (IT) relation ($IT \subseteq I \times I$, also written as '$\lesssim$') establishes issuer to IT 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 $i_r \lesssim i_e$, we call i_r the trustor and i_e the trustee. In MTAS model, trust is always established by the trustor allowing the trustee to view and use its own authorization statements. Therefore, the trustee can grant one of the trustor's roles, say r_r, a trustee's permission, say p_e. This role to permit assignment enables all users in r_r to inherit p_e. Further, the trustee can make one of the trustee's roles, say r_e, to be junior to one of the trustor's roles, say r_r. The effect of this role to role assignment is to make all users in r_r members of r_e so that the permissions of r_e in the trustee are also inherited by the users of r_r in the trustor. The definition of MTAS trust model is given as follows.

Definition 1
Let A and B denote two issuers. By establishing an IT relation with B ($A \lesssim B$), A exposes its entire RH 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 2.1, Bob, representing the resource owner E, could allow certain developers in OS to access the source code files stored in $Dev.E$ for them to conduct the out-sourcing job. Assume the proper permission in E for the out-sourcing job, *(edit, Dev.E, /src/)* is associated with 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 an appropriate developer role in OS, say $role(OS, dev)$. In this way, the users associated with $role(OS, dev)$ are able to edit the files under the */src/* directory in $Dev.E$.

The trust model solves the two key problems in collaborative RBAC: decentralized authority and semantic mismatch. Because 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 trustor exposes its authorization settings to the trustee, whereas the trustee assigns permissions of its data to the trustor. 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

[‡]The session component was not discussed in [9], 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.

communication with each other. In the trust model of MTAS, this issue is mitigated, because the authorization settings, that is, the RH and the role members, of the trustor are exposed to the trustee upon the creation of the IT relation. Consider the out-sourcing case. With the presence of $OS \precsim E$, E's administrator may examine the members of OS's roles and decide which role is appropriate to assign the permission to.

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);
- User ownership $(UO) \subseteq U \times I$, a many-to-one relation mapping each user to its owning issuer; correspondingly, $userOwner(u : U) \to I$, a derived function mapping a user to its issuer, where $userOwner(u) \in \{i \in I | (u, i) \in UO\}$;
- $RO \subseteq R \times I$, a many-to-one relation mapping each role to its owning issuer; correspondingly, $roleOwner(r : R) \to I$, 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, $permOwner(p : P) \to I$, 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 IT relation, also written as $\precsim$;
- $canUse(r : R) \to 2^I$, a derived function mapping a role to a set of issuers who can use the particular role. Formally, $canUse(r) = \{i \in I | roleOwner(r) \precsim i\}$;
- User assignment $(UA) \subseteq U \times R$, a many-to-many user-to-role assignment relation;
- Permission assignment $(PA) \subseteq P \times R$, 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 RH or role dominance relation, also written as $\geqslant$, requiring $r \geqslant r_1$, only if $roleOwner(r_1) \in canUse(r)$;
- $user(s : S) \to U$, a function mapping each session to a single user, which is constant within the lifetime of the session; and
- $roles(s : S) \to 2^R$, a function mapping each session to a subset of roles, $roles(s) \subseteq \{r | (\exists r' \geqslant r)[(user(s), r') \in UA \wedge userOwner(user(s)) \in canUse(r)]\}$, which can change within s, and s has the permissions $\bigcup_{r \in roles(s)} \{p | (\exists r'' \leqslant r)[(p, r'') \in PA]\}$.

Note that because we are formalizing an extension of pure RBAC model [10], the user PA described in [9] 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 RH, 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 National Institute of Standards and Technology (NIST) RBAC model [10]. In the RBAC96 model, implicit activation is specified [11]. 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.

Because every user identity is available to all the issuers, UA assignments bear no constraints on issuers. Thus, the UA assignments are always issued by the role owners as discussed in the administration of the MTAS model in Section 3.2.

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.

3.2. Administrative MTAS model

The administration model, AMTAS is tightly coupled with the MTAS model, because 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 AMTAS model is defined by the following two rules. Assume that A and B are two tenants and A trusts B.

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

As described in Definition 3, AMTAS maintains the balance of management by introducing 'dual control'. In any cross-issuer access, the resource requesting issuer controls the trust relations, which decide whether or not to allow cross-issuer assignments. The resource owner keeps the ultimate authority of its resources and issues the assignments based on properly created and maintained trust relations. Both the trust relations and the assignments are crucial in cross-issuer authorization because if either is revoked or altered, the corresponding collaborative accesses will be denied.

Table I provides the core logic of administrative functions of AMTAS. The functions are presented in a three-column format with function names, conditions, and updates. There are two parts of administrative functions available to two different levels of administration. The cloud administrators are roles empowered to add and remove issuers [§]. Along with the removal of an issuer, its correlated trust relations, users, roles, and permissions should also be removed. Even though the users are globally available, the removal of their owner issuers will result in removal of the users as well because the authentication of the users depends on their owner issuers. The removal of users will result in revocation of correlated UA. As cross-issuer UA is allowed, some assignments authorized by other issuers may also be removed. The same situation happens when an issuer is trying to remove one of its roles. In this way, cross-issuer authorization assignments are controlled by the resource owners, the permission owners in AMTAS.

The functions of assigning and revoking trust relations are controlled by the resource requesters. When a revocation of a trust relation is issued the trustor, the question of whether the correlated cross-issuer assignments (PA and RH) specified by the trustee is automatically removed or not is left as an implementation issue. For simplicity of our discussion, we choose the former. It is worth to note that the policy decision results are not affected by the choice because according to Definition 2, the authorization assignments will not function without the proper trust relation when the corresponding cross-issuer accesses are being checked.

3.3. Enhanced trust models

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

[§]Although in contemporary clouds, the administration commands tend to integrate self-service features without intervention by cloud administrators, they are also required to follow the built-in rules specified by the CSP.

Table I. Administrative functions in AMTAS

Function	Condition	Update
Administrative functions available to cloud administrators:		
AddIssuer(i)	$i \notin I$	$I' = I \cup \{i\}$
RemoveIssuer(i)	$i \in I$	**forall** $i_e \in I$ **do** RevokeTrust(i, i_e) RevokeTrust(i_e, i) **forall** $userOwner(u) \equiv i$ **do** RemoveUser(i, u) **forall** $roleOwner(r) \equiv i$ **do** RemoveRole(i, r) **forall** $permOwner(p) \equiv i$ **do** RemovePerm(i, p) $I' = I \setminus \{i\}$
Administrative functions available to issuer *i*:		
AddUser(i, u)	$userOwner(u) \equiv i \wedge u \notin U$	$U' = U \cup \{u\}$
RemoveUser(i, u)	$userOwner(u) \equiv i \wedge u \in U$	**forall** $\{r : R \mid (u, r) \in UA\}$ **do** RevokeUser(i, u, r) $U' = U \setminus \{u\}$
AddRole(i, r)	$i = roleOwner(r) \wedge r \notin R$	$R' = R \cup \{r\}$
RemoveRole(i, r)	$i = roleOwner(r) \wedge r \in R$	**forall** $\{u : U \mid (u, r) \in UA\}$ **do** RevokeUser(i, u, r) **forall** $\{p : P \mid (p, r) \in PA\}$ **do** RevokePerm(i, p, r) **forall** $\{r_{asc} : R \mid (r_{asc}, r) \in RH\}$ **do** RevokeRH(i, r_{asc}, r) **forall** $\{r_{desc} : R \mid (r, r_{desc}) \in RH\}$ **do** RevokeRH(i, r, r_{desc}) $R' = R \setminus \{r\}$
AddPerm(i, p)	$permOwner(p) \equiv i \wedge p \notin P$	$P' = P \cup \{p\}$
RemovePerm(i, p)	$permOwner(r) \equiv i \wedge p \in P$	**forall** $\{r : R \mid (p, r) \in PA\}$ **do** RevokePerm(i, p, r) $P' = P \setminus \{p\}$
AssignUser(i, u, r)	$i = roleOwner(r) \wedge u \in U$	$UA' = UA \cup \{(u, r)\}$
RevokeUser(i, u, r)	$i = roleOwner(r) \wedge u \in U \wedge (u, r) \in UA$	$UA' = UA \setminus \{(u, r)\}$
AssignPerm(i, p, r)	$i = permOwner(p) \wedge i \in canUse(r)$	$PA' = PA \cup \{(p, r)\}$
RevokePerm(i, p, r)	$i = permOwner(p) \wedge i \in canUse(r) \wedge (p, r) \in PA$	$PA' = PA \setminus \{(p, r)\}$
AssignRH(i, r_{asc}, r)	$i = roleOwner(r) \wedge i \in canUse(r_{asc}) \wedge \neg(r_{asc} \gg r) \wedge \neg(r \geqslant r_{asc})$ [†]	$\geqslant' = \geqslant \cup \{r_2, r_3 : R \mid r_2 \geqslant r_{asc} \wedge r \geqslant r_3 \wedge roleOwner(r_3) \in canUse(r_2) \bullet (r_2, r_3)\}$ [‡]
RevokeRH(i, r_{asc}, r)	$i = roleOwner(r) \wedge i \in$	$\geqslant' = (\gg \setminus \{(r_{asc}, r)\})^*$ [§] $canUse(r_{asc}) \wedge r_{asc} \gg r$
AssignTrust(i, i_e)	$i_e \in I$	$\precsim' = \precsim \cup \{(i, i_e)\}$
RevokeTrust(i, i_e)	$i_e \in I \wedge i \precsim i_e \wedge i \neq i_e$ [¶]	$\precsim' = \precsim \setminus \{(i, i_e)\}$ [♭]

[†] The notation '$\gg$' represents an immediate inheritance relation. This condition prevents cycle creation in the role hierarchy.
[‡] All the roles senior to r_{asc} become senior to all the roles junior to r.
[§] The notation '*' represents recursive updates for the entire role hierarchy.
[¶] An issuer cannot refuse to trust itself. Otherwise, improper revocation of assignments may occur.
[♭] By revoking the trust relation, the canUse() function of i's roles automatically updates accordingly as well as PA and RH.

3.3.1. Trustor-centric public role. As the name suggests, TCPR introduces the public role constraint for trustors. The public roles are included in a predefined subset of a trustor's roles exposed to all of the trustees. It is formally defined as follows.

Definition 4
The 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 is expose to its trustees; and
- $canUse(r : R) \rightarrow 2^T$ is modified to $canUse(r) = \{i\} \cup \{i_1 \in i | i \precsim i_1 \wedge r \in \mathcal{P}_T(i)\}$, 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 .

Because the public roles in TCPR are defined in terms of the trustor i, if $\mathcal{P}_T(i)$ is modified, then all the trust relations with the common trustor 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.

3.3.2. Relation-centric public role. In contrast with TCPR, RCPR enforces the public role constraints for trust relations instead of trustors. The public roles are included in a predefined subset of the trustor's roles exposed to the trustee in a specific trust relation. The formal definition follows.

Definition 5
The RCPR model inherits all the components from MTAS in Definition 2, while the following modifications are applied:

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

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

3.4. Constraints

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

Cyclic RH. A 'role cycle' may be formed across tenants in MTAS systems without proper constraints. This may lead to violation of the security principal [32] in interoperation.

Similar problems in secure interoperation have been addressed in multi-domain environment [33]. Some computational challenges discussed in [32] remain in the multi-tenant cloud environment. 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-issuer 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 the *AssignRH* function in AMTAS includes these provisions.

Separation of duties (SoD). During collaborations with MTAS, we identify two levels of SoD, issuer level and role level. For issuer level SoD, one collaborating issuer cannot execute two conflict responsibilities. For instance, SOX [34] 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 out-sourcing example as shown in Figure 1, 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. In more general cases, the two roles with conflict duties may come from different tenants. Also, the conflict duties may be associated with different cloud services. In this case, the constraint policy composition process can become very complicated [35]. Thus, the CSP should be responsible to identify potential conflict roles. Each tenant should be aware of the SoD problems and specify proper constraint policies in the reference with the conflict roles.

Chinese wall. The conflict of interests (COIs) among issuers also need 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 [36], which can be integrated in the centralized AaaS platform to avoid COIs. Essentially, the issuers are grouped into 'COI classes', and by mandatory ruling, all issuers are allowed to trust at most one issuer belonging to each such COI class. The CSP or a third party authority should be responsible to maintain the COI classes. In this way, no cross-issuer access will be assigned or permitted by the other COI issuers.

3.5. Trusts in Amazon Web Service (AWS) and OpenStack

As needs of multi-tenant collaboration keep growing, more and more cloud software vendors are trying to establish cross-tenant or cross-issuer access control mechanisms. Amazon, as one of the biggest public CSPs, allows its user to establish trust between two accounts in AWS in order to simplify user management between Production and Development accounts and many other use cases [37]. Account is a counterpart of issuer in MTAS. Moreover, the unilateral trust relation between accounts is very similar to the trust relation between issuers but the types are different. Take the Production and Development accounts use case for example. In order to allow cross-account accesses in AWS, the Production account, the trustor, establishes a trust relation with the Development account, the trustee, first. Then the Production account specifies an authorization assignment allowing only developers, as one of the roles, from the Development account to access some of its resources. As described earlier, the AWS account trust does not provide dual control feature, and some offline communications, such as the role names, are required. In MTAS, the trust relation is established by the Development account so that the Production account can see its roles, users, and assignments and make cross-account authorization assignments based on the information. In this way, offline communication is avoided, and the Development account can control cross-account authorization with trusts.

OpenStack is an open source cloud computing platform for public and private clouds [38]. The trust mechanism inside OpenStack is built on the basis of user-level delegation. Use the same Production and Development accounts as examples. The concept of account or issuer is close to the concept of domain in OpenStack. It provides administrative boundary for users and projects. In OpenStack, a user Paul in Production domain can set up a trust relation with another user David in Development domain. The trustee user David can inherit a subset of Paul's roles and access project resources in the Production. The collaboration is enabled and fully controlled by Paul. This trust relation is flexible and easy to achieve collaboration. However, improper use of this trust may result in breaches of security boundaries between domains. If MTAS is integrated in OpenStack, domain administrators will be in charge of maintaining the trust relations and authorization assignments. Hence, the risk of access control breaches caused by user behavior can be lowered. Again, dual control mechanism is introduced between domain administrators.

In fact, there are some other types of trusts to choose from [39–41]. Different types of trust relations may suit different needs for collaboration. However, a consolidated access control model for cross-tenant collaboration in the real world cloud is still not generally available.

4. POLICY SPECIFICATION

In order to demonstrate the feasibility of the MTAS model, we give the policy specification here in XACML. The normative specification of RBAC policies with XACML2.0 language has been proposed by OASIS XACML TC [42]. Its Role PolicySet (RPS) and Permission PolicySet (PPS), representing UA and PA, respectively, are also used with the MTAS policy specification. Additionally, a novel Trust PolicySet (TPS) is added to express the trust relation.

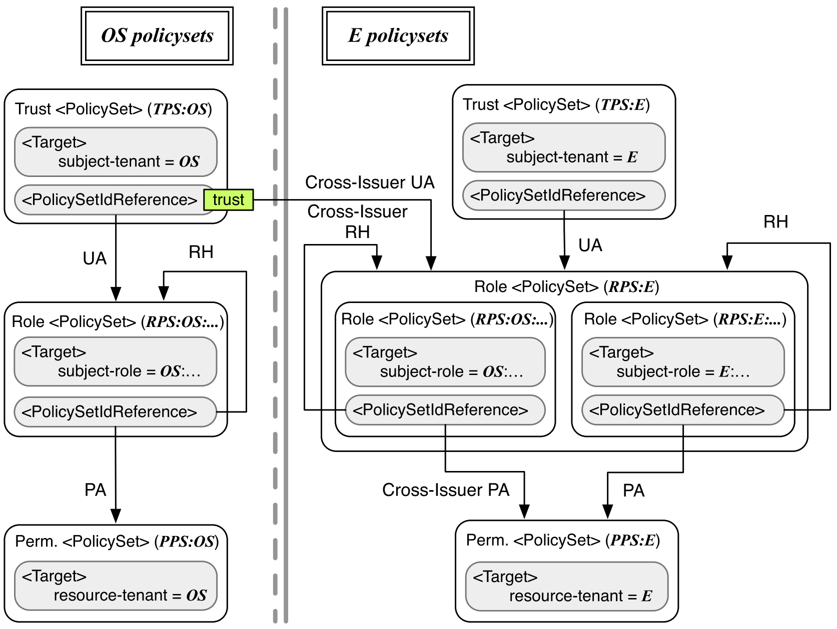

OS policysets
E policysets
Trust <PolicySet> (TPS:OS)
<Target> subject-tenant = OS
<PolicySetIdReference>
trust
UA
RH
Role <PolicySet> (RPS:OS:...)
<Target> subject-role = OS:...
<PolicySetIdReference>
PA
Perm. <PolicySet> (PPS:OS)
<Target> resource-tenant = OS
Cross-Issuer UA
Cross-Issuer RH
Trust <PolicySet> (TPS:E)
<Target> subject-tenant = E
<PolicySetIdReference>
UA
RH
Role <PolicySet> (RPS:E)
Role <PolicySet> (RPS:OS:...)
<Target> subject-role = OS:...
<PolicySetIdReference>
Role <PolicySet> (RPS:E:...)
<Target> subject-role = E:...
<PolicySetIdReference>
Cross-Issuer PA
PA
Perm. <PolicySet> (PPS:E)
<Target> resource-tenant = E

Figure 3. Example MTAS policy structure with trust relation $OS \lesssim E$ as highlighted.

To better explain how MTAS XACML policy work, we develop an example policy structure as shown in Figure 3 consistent with the out-sourcing example as described in Section 2.1. For instance, Charlie as a user in OS with a manager role in E, namely, E:*manager*, requests to access a permission to create a repository in E, namely, $E{:}cr$. Following the convention in XACML, we use ':' as the delimiter of namespaces. At the PDP end, OS's TPS, written as $TPS{:}OS$, states $OS \lesssim E$, which is simply adding $RPS{:}E$ as a referenced PolicySet. E's RPS, written as $RPS{:}E$, states that E:*manager* dominates E:*employee*, which is the *employee* role in E. Meanwhile, E's PPS, written as $PPS{:}E$, states that E:*employee* is permitted to have the permission $E{:}cr$. As a start, the request is sent from PEP to PDP where $TPS{:}OS$ is invoked because the subject tenant attributes in the request valued as OS. According to the MTAS, as long as the trust relation $OS \lesssim E$ exists, $TPS{:}OS$ is able to reference both $RPS{:}OS$ and $RPS{:}E$. Thus, the request is forwarded to both. A dead end will be reached in $RPS{:}OS$ because the requested permission is in E. In $RPS{:}E$, the request is forwarded to $RPS{:}E$:*manager* and then to $RPS{:}E$:*employee* due to the RH assignment. The PDP will probe into the referenced $PPS{:}E$ policies from $RPS{:}E$ along the process and find a match in $PPS{:}E{:}cr$. Then, a permit decision will be responded to PEP who will lead Charlie the requested access.

If Charlie uses another role OS:*manager* requesting the same permission, the authorization path is different. At the PDP end, E states that OS:*manager* dominates E:*employee*, and OS:*manager* has permission $E{:}cr$. When the request is forwarded to $RPS{:}E$ through the same aforementioned trust relation, PDP will look into the policies inside $RPS{:}E$. The $RPS{:}OS$:*manager* will be reached and will forward the request to both $PPS{:}E{:}cr$ and $RPS{:}E$:*employee*. Both paths representing cross-issuer PA and RH, respectively, will return a permit for the request.

In this example, we can clearly see that cross-issuer accesses are properly controlled by MTAS policies. The policy specification could be directly used in MTAS implementation.

5. PROTOTYPE IMPLEMENTATION

In order to further explore the feasibility of MTAS, we developed an AaaS prototype system using SUN's XACML implementation [43] with respect to OASIS standards [44]. The experiments are designed to benchmark the prototype performance and scalability in the cloud environment.

5.1. Implementation details

The prototype system has a centralized PDP and multiple distributed policy enforcement points (PEPs), which are in charge of forwarding access requests to the PDP and enforcing authorization decisions for corresponding cloud services. Because each service is built-in with multi-tenancy, the MTAS model will aggregate the permissions in various services for each tenant. In practice, the PDP can be built in fully distributed manner to achieve better performance. However, the policy discovery algorithms in a distributed environment are beyond the scope of our discussion, and the policy discovery latency is unpredictable because of various factors in implementation. Thus, a centralized deployment is better to show the metrics for experiment purpose.

The PDP and PEP modules are compiled, deployed, and evaluated on virtual machines (VMs) created in a private cloud system running Joyent Cloud [45]. The PDP is installed on a 64-bit Linux CentOS 6 system with 2.5 GHz dedicated CPUs. The PEPs are built upon SmartMachines [45] with SmartOS 1.8.1, 256-MB RAM and shared CPUs.

SmartMachine CPU caps are set to 350 meaning each can use 3.5 CPUs in maximum. The VMs for PDPs and PEPs are deployed in different security zones with different networks and physical racks so that the performance evaluation results are not affected by virtualization level interference. All the machines in the prototype are connected through data center networks. The architecture of our testbed is described in Figure 4. We used eight VMs with the same flavor to send concurrent requests to the centralized PDP, which is deployed on various testing machines with different capacities for scalability tests. The automated test controller (ATC) synchronizes the code of the system with all the VMs, runs the PDP service on a particular testing server, and configures the PEPs to send all the requests to the server. The experiment results are also collected by the ATC. The testbed architecturally simulates an authorization service implementing the MTAS model and supports controlled experiments in a commercial-standard cloud system.

5.2. Experiments and results

In order to measure the scalability of PDPs, we define a computing capacity unit as 1-GB RAM and 1 Core CPU. Because standard commodity hardware dominates the cloud, VM CPU frequencies are usually identical with each other in the same environment. Thus, the number of CPU cores, rather than value of CPU frequency, is regarded as proportional to hardware capability. For example, a PDP with 2 GB RAM and 2 Core CPU is considered as of 2-unit computing capacity, which doubles hardware capability of its 1-unit counterpart. In our experiments, we have PDPs running on 1-unit, 2-unit, and 4-unit servers, respectively. At the PEP end, we have eight SmartMachines of the same capacity to generate authorization requests so that the volumes of PEP requests can be scaled proportionally with the PDP capacity.

Performance. Policy decision latency is one of the most important metrics in performance evaluation of access control systems. An MTAS decision process consists of several procedures: subject and resource verification, attribute searching, and retrieving referenced policy files. These procedures take policy decision time at the PDP end. We call this effect authorization overhead, which is inevitable.

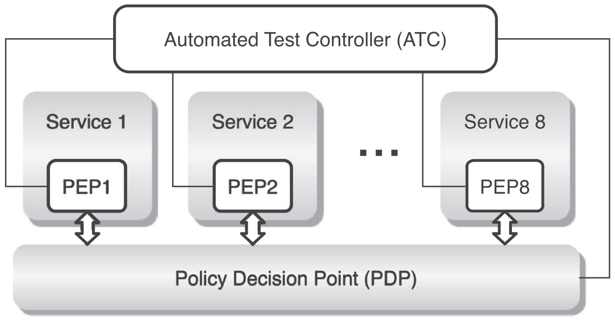
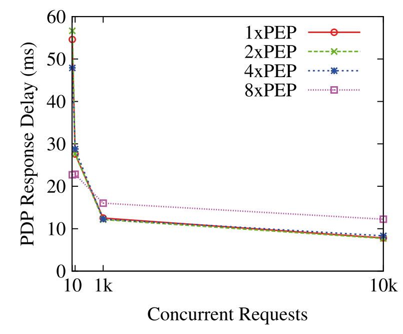
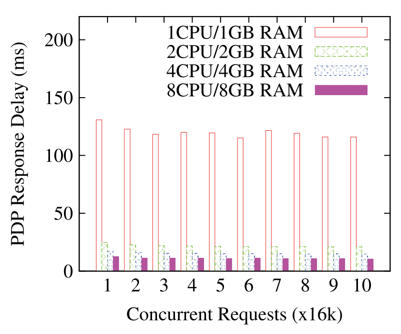


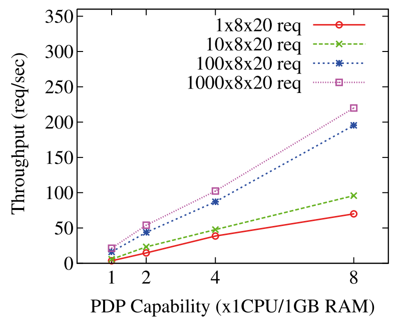
Figure 4. MTAS Testbed Architecture.



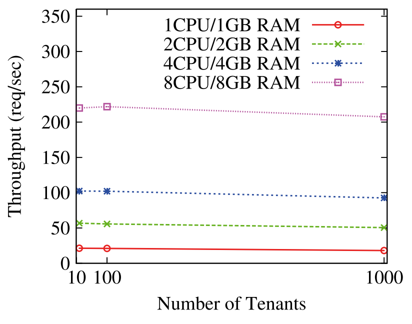
(a) PDP Response Delay with various PEP amount



(b) PDP Response Delay with various hardware capability



(c) Scalability Results with 10 tenants



(d) Policy Complexity Scalability Results

Figure 5. Performance and scalability evaluation results.

We evaluate the authorization overhead of our prototype system by sending concurrent sample requests from various numbers of PEPs to a 4-unit PDP. There are 10 disjoint pairs of sample requests and responses representing intra-issuer and cross-issuer accesses. The horizontal axis represents the count of concurrent requests sent by PEPs, and the vertical axis shows the average PDP response delay measured at the PEP end. At the beginning, the average response time falls steeply as concurrent requests from each PEP increase. Because the policy files are loaded at run time at the PDP end, it takes longer time to respond for the first requests and then as the caching mechanisms of the operating system function, the average response latency tends to reach a stable state. For our prototype system, as shown in Figure 5(a), the average PDP response delay at the stable state is around 12 ms, which is acceptable for ordinary deployment in the cloud.

The PDP hardware capacity is also an important factor of PDP response delay. Figure 5(b) illustrates the performance of PDP on different servers with 1, 2, 4, and 8 units of hardware capacity, respectively. The experiments are conducted with eight PEPs and 1000 tenants. The result shows that with the same hardware capacity, the response delay is relatively stable. It is reasonable that more powerful PDP causes less response delay which drops around 80% between 1-unit and 2-unit PDPs. This outcome leads us to test the throughput scalability of the prototype system.

Scalability. Dynamic scaling is one of the key features of cloud computing. Authorization mechanisms in the cloud also need to be scalable. We evaluate the scalability of our prototype from both capacity and policy complexity view points. A scalable system should have its performance improvement proportional to the hardware capacity increase. In the mean time, the complexity of policy also influences the system performance. In our experiments, we identify that the number of tenants is the major factor in the authorization overhead. Thus, we also measure the scalability by increasing the number of tenants in orders of magnitude.

The capacity scalability evaluation compares the authorization overhead of PDPs with various computing capacity units. The throughput of authorization requests is calculated using the following formula.

$$Throughput = \frac{1}{Average\_Delay \times CPU\_Utilization} \tag{5}$$

The results shown in Figure 5(c) give a clear view that the speedup of PDP servers increases the throughput proportionally. The result is validated against multiple scales of concurrent requests.

The policy complexity scalability evaluation takes the number of tenants into account to measure the trend of how the policy complexity affects the performance of the system. Figure 5(d) plots the results with the number of tenants on the x-axis and the authorization overhead on the y-axis. In the experiment, the total concurrent requests number is 160,000 which is fairly dispersed to all the tenants. The trend shows that the increase of tenants does not cause steep drop of the system performance and is inversely proportional with the throughput. Consequently, it is reasonable to believe that the prototype is scalable in the cloud environment.

6. RELATED WORK

The RBAC [10, 11] model provides efficient authorization solutions within enterprises and organizations. Many RBAC extensions have been proposed to address access control problems among multiple organizations. Some extensions [13, 15] introduce centralized authority to specify or mediate cross-domain policies. However, in cloud environment, the only existing centralized authority is the CSP who should maintain the generic policies for all the tenants rather than for specific ones because the tenants are temporary and self-service oriented. Thus, these approaches are not directly applicable in the cloud. Some other extensions use decentralized authorities to RBAC using delegation [8, 18]. In particular, a user may delegate the assigned permissions, entirely or partially, to other users. The delegation is maintained by the delegator who can specify whether the delegation is transitive or not. The delegation approach requires additional administration. All the delegations of a permission need to be kept track of well in a graph. If any of the nodes, typically users, change, the entire graph of authorization will change unexpectedly. Hence, the complexity of the delegation approaches becomes overwhelming in the agile cloud environment.

Addressing multi-domain secure interoperation, the role mapping approaches [33, 35, 46] allow external users in trusted domains to acquire privileges in a local domain through role hierarchy relations. These approaches focus on extending RBAC models and enabling interoperation by mapping roles between domains. Shafiq *et al.* [33] address the issue of optimal RBAC policy resolution among multiple domains by an integer-programming-based approach. It helps maintain autonomy for inter-domain role accesses. Baracaldo *et al.* [35] present a policy enforcement framework to enforce temporal and SoD constraints for multi-domain interoperation. Moreover, Shehab *et al.* [46] introduce a distributed secure interoperability framework to link multiple domains. The access path discovery algorithm is critical to enforce secure accesses in this framework. The problems in multi-domain scenarios are similar to those in multi-tenant scenarios so that the role mapping approaches are potentially applicable in the cloud. But the role mapping approaches and MTAS are solving problems in different layers. The role mapping approaches focus on linking and discovery algorithms to connect roles between domains based on RBAC. Yet, MTAS focuses on extending RBAC with multi-tenant features and facilitating collaborations with trust relations between tenants. Unlike the role-mapping approaches, the trust relation in MTAS is not transitive so that the access paths never exceed the range of two tenants which is the normal case in the cloud because each tenant owns the ultimate authority of its resources. Also, MTAS allows not only cross-tenant role hierarchy assignments but also direct cross-tenant permission assignments supported by the homogeneous architecture of the cloud infrastructure.

Federated identity and authorization services are included in the architecture of distributed systems to facilitate collaborative access control. Federated identity [20, 47] enables authenticating strangers by sharing identity information among federated parties. The federation relation is intrinsically an equal trust relation, which is meaningful in some used cases but cumbersome in supporting the variety of collaborations. For example, the trust relation between E and OS in Figure 1 is apparently not an equal trust because the ultimate owner of the resources to be accessed is E but not both of them. Moreover, the maintenance of 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 [21, 23, 25] aim at secure resource sharing among virtual organizations in grid computing systems leveraging cryptographic credentials. Although such credential-driven approaches are viable and effective, the overhead of maintaining the public-key infrastructure for credentials is expensive and not necessary in the cloud environment with the existence of centralized facility and homogeneous architecture.

Along with the development of cloud computing, multi-tenant authorization, as an enabling technology, is being researched in both the industry and the academia. Calero *et al*. [9] propose a centralized multi-tenancy authorization system for cloud services. It bridges RBAC systems using a trust model which is coarse-grained and open for extensions. Continuing their work, MTAS [39] formalizes the authorization model and extends it with finer-grained trust relations in terms of roles exposed to trustee tenants. MT-RBAC [40] is similar but uses a different trust model suitable for scenarios requiring the trustor to partially delegate the authority of resources to the trustees. CTTM [41] gives a systematic taxonomy of useful trust relations between tenants. The trust types cover RT [7], MTAS, and MT-RBAC. OSAC-DT [48] implements the CTTM models in a domain trust module into Keystone, the identity service of OpenStack cloud system. The dynamic nature of multi-tenancy distinguishes the access control model with the traditional ones. It is the key to culture secure collaboration in the cloud.

7. CONCLUSION AND FUTURE WORK

To support collaboration between cloud services, we formalize an MTAS model based on an informally specified MTAS [9], which extends the RBAC model by building trust relations among collaborating services. Further, we give the AMTAS and enhancements (TCPR and RCPR) 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. Further, we implement a prototype MTAS system in a cloud and evaluate its performance and scalability. The results provide more insights and confidence of MTAS to be integrated with collaborative cloud services.

Currently, our research team is working toward various collaborative access control models, both role-based and attribute-based, using similar trust relations. Further research in feasible trust models, and generic trust frameworks are anticipated to emerge from this line of research. Also, in order to explore the applicability of the multi-tenant access control models, they will be integrated with real world cloud systems in our future work.

ACKNOWLEDGEMENTS

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

REFERENCES

1. Singhal M, Chandrasekhar S, Ge T, Sandhu R, Krishnan R, Ahn G-J, Bertino E. Collaboration in multicloud computing environments: framework and security issues. *IEEE Computer* 2013; **46**(2):76–84.
2. Mell P, Grance T. The NIST definition of cloud computing. Special Publication 800-145, 2011.
3. Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, Lee G, Patterson DA, Rabkin A, Stoica I, Zaharia M. Above the clouds: a Berkeley view of cloud computing. *Technical Report*, EECS Department, University of California, Berkeley, 2009.
4. McKenty J. Nebula's implementation of role based access control (RBAC). (Available from: http://nebula.nasa.gov/blog/2010/06/03/nebulas-implementation-role-based-access-control-rbac/) [Accessed on 3 June 2010].
5. Chong RF. Designing a database for multi-tenancy on the cloud. (Available from: http://www.ibm.com/developerworks/data/library/techarticle/dm-1201dbdesigncloud/index.html) [Accessed on 26 January 2012].
6. Chong F, Carraro G, Wolter R. Multi-tenant data architecture. (Available from: http://msdn.microsoft.com/en-us/library/aa479086.aspx) [Accessed on June 2006].
7. Li N, Mitchell JC, Winsborough WH. Design of a role-based trust-management framework. *Proceedings of the 2002 IEEE Symposium on Security and Privacy,* IEEE, Oakland, California, USA, 2002; 114–130.

8. Freudenthal E, Pesin T, Port L, Keenan E, Karamcheti V. dRBAC: distributed role-based access control for dynamic coalition environments. *Proceedings of the 22nd International Conference on Distributed Computing Systems,* IEEE, Vienna, Austria, 2002; 411–420.
9. Calero JMA, Edwards N, Kirschnick J, Wilcock L, Wray M. Toward a multi-tenancy authorization system for cloud services. *Security Privacy, IEEE* 2010; **Nov/Dec 2010**:48–55.
10. Ferraiolo DF, Sandhu R, Gavrila S, Kuhn DR, Chandramouli R. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)* 2001; **4**(3):224–274.
11. Sandhu RS, Coyne EJ, Feinstein HL, Youman Charles E. Role-based access control models. *IEEE Computer* 1996; **29**(2):38–47.
12. Cohen E, Thomas RK, Winsborough W, Shands D. Models for coalition-based access control (CBAC). *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT),* ACM, 2002; 97–106.
13. Li Qi, Zhang Xinwen, Xu Mingwei, Wu Jianping. Towards secure dynamic collaborations with group-based RBAC model. *Computers & Security* 2009; **28**(5):260–275.
14. Lin D, Rao P, Bertino E, Li N, Lobo J. Policy decomposition for collaborative access control. *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT),* ACM, Estes Park, Colorado, USA, 2008; 103–112.
15. Zhang Z, Zhang X, Sandhu R. ROBAC: scalable role and organization based access control models. *Proceedings of the 2006 International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom),* IEEE, Atlanta, Georgia, USA, 2006; 1–9.
16. Alam M, Zhang X, Khan K, Ali G. xDAuth: a scalable and lightweight framework for cross domain access control and delegation. *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT),* ACM, Innsbruck, Austria, 2011; 31–40.
17. Bauer L, Jia L, Reiter MK, Swasey D. xDomain: cross-border proofs of access. *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT),* ACM, Stresa, Italy, 2009; 43–52.
18. Zhang X, Oh S, Sandhu R. PBDM: a flexible delegation model in RBAC. *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT),* ACM, Villa Gallia, Como, Italy, 2003; 149–157.
19. Barka E, Sandhu R. Framework for role-based delegation models. *Proceedings of the 16th Annual Conference on Computer Security Applications (ACSAC),* IEEE, New Orleans, Louisiana, USA, 2000; 168–176.
20. Bhatti R, Bertino E, Ghafoor A. An integrated approach to federated identity and privilege management in open systems. *Communications of the ACM* 2007; **50**(2):81–87.
21. Alfieri R, Cecchini R, Ciaschini V, dell'Agnello L, Frohner Á, Lőrentey K, Spataro F. From gridmap-file to VOMS: managing authorization in a grid environment. *Future Generation Computer Systems* 2005; **21**(4):549–558.
22. Bertino E, Mazzoleni P, Crispo B, Sivasubramanian S. Towards supporting fine-grained access control for grid resources. *Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS),* IEEE, Suzhou, China, 2004; 59–65.
23. Chadwick DW, Otenko A. *The PERMIS X. 509 Role Based Privilege Management Infrastructure*, Vol. 19. Elsevier: Amsterdam, Netherlands, 2003, 277–289.
24. Mazzoleni P, Crispo B, Sivasubramanian S, Bertino E. Efficient integration of fine-grained access control and resource brokering in grid. *The Journal of Supercomputing* 2009; **49**(1):108–126.
25. Pearlman L, Welch V, Foster I, Kesselman C, Tuecke S. A community authorization service for group collaboration. *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks,* IEEE, Monterey, California, USA, 2002; 50–59.
26. Foster I, Zhao Y, Raicu I, Lu S. Cloud computing and grid computing 360-degree compared. *Grid Computing Environments Workshop (GCE),* IEEE, Austin, Texas, USA, 2008; 1–10.
27. Adams AK, Lee AJ, Mossé D. Receipt-mode trust negotiation: efficient authorization through outsourced interactions. *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security,* ACM, Hong Kong, 2011; 430–434.
28. Jin J, Ahn G-J. Role-based access management for ad-hoc collaborative sharing. *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies (SACMAT),* ACM, Lake Tahoe, California, USA, 2006; 200–209.
29. Jin J, Ahn G-J, Shehab M, Hu H. Towards trust-aware access management for ad-hoc collaborations. *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom),* IEEE, New York, USA, 2007; 41–48.
30. Bhatti R, Bertino E, Ghafoor A. X-FEDERATE: a policy engineering framework for federated access management. *IEEE Transactions on Software Engineering* 2006; **32**(5):330–346.
31. Chadwick D, Zhao G, Otenko S, Laborde R, Su L, Nguyen TA. Permis: a modular authorization infrastructure. *Concurrency and Computation: Practice and Experience* 2008; **20**(11):1341–1357.
32. Gong L, Qian X. Computational issues in secure interoperation. *IEEE Transactions on Software Engineering* 1996; **22**(1):43–52.
33. Shafiq B, Joshi JB, Bertino E, Ghafoor A. Secure interoperation in a multidomain environment employing RBAC policies. *IEEE Transactions on Knowledge and Data Engineering* 2005; **17**(11):1557–1577.
34. Sarbanes-Oxley Act (SOX). U.S. Public Law 107-204, 2002.
35. Baracaldo N, Masoumzadeh A, Joshi J. A secure, constraint-aware role-based access control interoperation framework. *Proceedings of the 5th Iinternational Conference on Network and System Security (NSS),* IEEE, Milan, Italy, 2011; 200–207.

36. Brewer DF, Nash MJ. The Chinese wall security policy. *Proceedings of the 1989 IEEE Symposium on Security and Privacy,* IEEE, Oakland, California, USA, 1989; 206–214.
37. Walkthrough: cross-account api access using IAM roles. (Available from: http://docs.aws.amazon.com/IAM/latest/UserGuide/cross-acct-access-walkthrough.html) [Accessed on 8 May 2010].
38. OpenStack. (Available from: http://www.openstack.org/) [Accessed on 17 October 2013].
39. Tang B, Sandhu R, Li Q. Multi-tenancy authorization models for collaborative cloud services. *Proceedings of the 14th International Conference on Collaboration Technologies and Systems (CTS),* IEEE, San Diego, California, USA, 2013; 132–138.
40. Tang B, Li Q, Sandhu R. A multi-tenant RBAC model for collaborative cloud services. *Proceedings of the 11th IEEE Conference on Privacy, Security and Trust (PST)* IEEE, Tarragona, Spain, 2013; 229–238.
41. Tang B, Sandhu R. Cross-tenant trust models in cloud computing. *Proceedings of the 14th IEEE Conference on Information Reuse and Integration (IRI)* IEEE, San Francisco, USA, 2013; 129–136.
42. Core and hierarchical role based access control (RBAC) profile of XACML v2.0. OASIS Standard, 2005.
43. Sun's XACML implementation. (Available from: http://sunxacml.sourceforge.net/) [Accessed on 16 July 2004].
44. OASIS eXtensible Access Control Markup Language (XACML) v2.0 specification set, 2005. (Available from: http://www.oasis-open.org/committees/xacml/) [Accessed on 1 February 2005].
45. Joyent SmartOS. (Available From: http://smartos.org/) [Accessed on 19 September 2013].
46. Shehab M, Bertino E, Ghafoor A. SERAT: SEcure role mApping technique for decentralized secure interoperability. *Proceedings of the tenth ACM Symposium on Access Control Models and Technologies (SACMAT)* ACM, Stockholm, Sweden, 2005; 159–167.
47. Chadwick D. Federated identity management. In *Foundations of Security Analysis and Design V*, vol. 5705, Lecture Notes in Computer Science. Springer: Berlin Heidelberg, 2009; 96–120.
48. Tang B, Sandhu R. Extending openstack access control with domain trust. *Proceedings of the 8th International Conference on Network and System Security (NSS)* Springer, Xi'an, China, 2014; 54–69.