MT-ABAC: A Multi-Tenant Attribute-Based Access Control Model with Tenant Trust

Navid Pustchi(✉) and Ravi Sandhu

Institute for Cyber Security, Department of Computer Science,
University of Texas San Antonio, One UTSA Circle, San Antonio, TX 78249, USA
tam498@my.utsa.edu, ravi.sandhu@utsa.edu

Abstract. A major barrier to the adoption of cloud Infrastructure-as-a-Service (IaaS) is collaboration, where multiple tenants engage in collaborative tasks requiring resources to be shared across tenant boundaries. Currently, cloud IaaS providers focus on multi-tenant isolation, and offer limited or no cross-tenant access capabilities in their IaaS APIs. In this paper, we present a novel attribute-based access control (ABAC) model to enable collaboration between tenants in a cloud IaaS, as well as more generally. Our approach allows cross-tenant attribute assignment to provide access to shared resources across tenants. Particularly, our tenant-trust authorizes a trustee tenant to assign its attributes to users from a trustor tenant, enabling access to the trustee tenant's resources. We designate our multi-tenant attribute-based access control model as MT-ABAC. Previously, a multi-tenant role-based access control (MT-RBAC) model has been defined in the literature wherein a trustee tenant can assign its roles to users from a trustor tenant. We demonstrate that MT-ABAC can be configured to enforce MT-RBAC thus subsuming it as a special case.

Keywords: Attribute-based access control · Distributed access control · Multi-tenant · Authorization federation · Security

1 Introduction

Cloud computing has dramatically altered the delivery of IT infrastructure and resources to organizations. Characteristics such as on-demand self service and resource pooling, provide flexibility and dynamicity at scale for cloud service consumers [16]. The benefits of cloud computing have been well documented in the literature and proven in the marketplace.

Cloud service providers (CSPs) segregate the resources and customer's data into tenants to protect data privacy and integrity. Tenants are isolated containers with tenant-specific virtual computing environments. Each tenant corresponds to an organization, a department of an organization, or an individual who uses cloud services. In this scenario, each tenant is considered as a cloud customer with resources whose integrity and privacy must be protected. The focus on tenant isolation diminishes the scope for collaboration across tenants.

M. Qiu et al. (Eds.): NSS 2015, LNCS 9408, pp. 1–15, 2015.
DOI: 10.1007/978-3-319-25645-0_14

At the dawn of cloud systems, the multi-tenancy concern was resource segregation, whereas recent enterprise cloud adoption has raised the issue of multi-tenancy resource sharing. The drive for multi-tenant collaboration arises from at least two distinct directions. First, a large organization may utilize multiple tenants for security and reliability, where each tenant can represent a department. For example, an organization's financial department processes sensitive financial data while its marketing department publishes open information to the public, so they need to be isolated but yet may need controlled collaboration. Second, distinct enterprises may have collaborative tasks across their corresponding tenants. Current cloud Infrastructure-as-a-service (IaaS) providers such as Amazon EC2 [1] or OpenStack [2] offer limited or no cross-tenant access [14].

In this paper we present a novel attribute-based access control model to enable collaboration between tenants in cloud systems. Our scope is limited to cross-tenant collaboration in a single cloud. This allows us to focus more on collaborative access control models, and defer consideration of cross-cloud integration issues.

To motivate the problem, consider the example illustrated in Figure 1, which depicts an organization with multiple tenants in a cloud service provider. We use HP as an organization with multiple locations and departments. In such organizations it is not feasible to locate all data and users into one tenant due to different security and reliability levels required as well as management barriers. Also, adding accounts for users across each collaborating tenant is impractical.

A practical approach for the cloud service provider is to support collaboration mechanisms across trusted tenants. Users in one tenant can access resources in another tenant consistent with cross-tenant trust relationships. It is natural for software development, testing, and support teams to collaborate. Software developers such as Alice can access cross-tenant resources in Software Testing and Software Support tenants to perform their assigned tasks. Enabling seamless collaboration across tenants is essential for the overall organization. Similar scenarios arise for cross-organization collaboration.

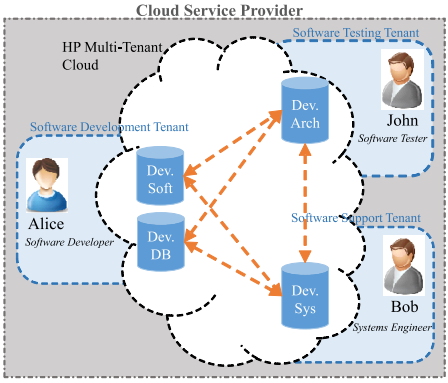

Fig. 1. A Multi-Tenant Collaboration Example

Current cloud IaaS providers such as Amazon or Rackspace provide intra-tenant access control using variations of the well-known role-based access control (RBAC) [6,20] approach. In RBAC access to a resource is based upon role-membership of the requesting user and resource-permission.

The notion of multi-tenant RBAC has been proposed to support multi-tenant collaboration in single cloud [24,26] or multi-cloud [17] environments.

RBAC has been the dominant access-control paradigm for over two decades. Nevertheless, various limitations of RBAC have been recognized over this period and increasingly there is a push to move towards attribute-based access control [9,10,18] in general. ABAC advantages over RBAC specifically in cloud computing have been discussed in the literature [5]. A user's access to a resource in ABAC depends on the relative values of the user and resource attributes. An attribute is simply a name:value pair. Attributes are used to represent security-relevant properties of users and resources. We anticipate that CSPs will incorporate ABAC features in addition to their currently implemented RBAC.

Our contribution in this paper is to develop a multi-tenant ABAC model with cross-tenant trust. To our knowledge this is the first work to consider cross-tenant attribute assignment in ABAC in a multi-tenant context.

The remainder of this paper is organized as follows. Section 2 introduces our core ABAC model entities and attribute functions in a single tenant model. In Section 3, our multi-tenant ABAC (MT-ABAC) model is proposed and specified. In section 4, we review the multi-tenant RBAC model from the literature and demonstrate how it can be configured in MT-ABAC. Section 5 discusses related work and section 6 gives our conclusions.

2 $ABAC_0$ Model

In this section, we present our core ABAC model which we designate as $ABAC_0$. This model is designed to be sufficient for our purpose in developing MT-ABAC and is not intended to be a comprehensive ABAC model. ABAC has been defined in various ways in the literature, usually for some specific purpose. Our model is specifically motivated by the previously defined $ABAC_\alpha$ model [11] and is compatible with the recently defined NIST ABAC framework [9].

Core $ABAC_0$ model element sets and functions are illustrated in Figure 2, which includes three basic components: users (U), objects (O), and actions (A). Attributes are properties associated with users and objects which we represent by $UATT$ and $OATT$ respectively. Users and objects are collectively called entities. Authorization predicates ($Auth$) express access rules in the system which evaluate user attributes against object attributes and render a decision to permit or deny access to the requested resource with respect to the specific action.

Each *attribute* is a function which takes users or objects as input and returns a value from the attribute's range (we use the terms range and scope interchangeably). For example, a user attribute function such as $Role \in UATT$ maps $u_1 \in U$ to a value $cloud\_admin$. Depending upon attribute type each attribute function will return a single value or a set of values. An atomic-valued attribute will return one value while a set-valued attribute will return a subset of values within its defined scope.

A user can be a human or non-person entity, such as an application, making requests to perform actions on an object. We consider a user ($u \in U$) to be a person for simplicity. Each user is represented by a finite set of user attributes ($UATT$) such as name, salary, clearance, role, etc. User attribute function values are specified by security architects at system creation or modification time.

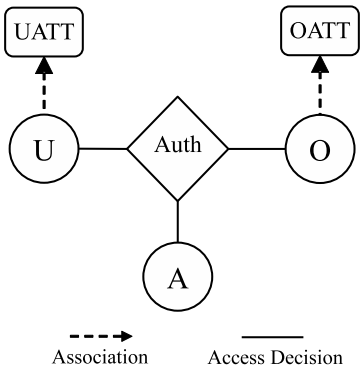

Fig. 2. Core $ABAC_0$ Model Structure.

Objects are system resources for which access should be protected such as files, applications, virtual machines (VMs), etc. Objects are associated with attribute functions ($OATT$) representing resource properties such as risk level, location, and classification. At creation or modification time object attributes might be constrained by the attributes of creating user in the system, for example, a new VM object can inherit attributes such as VM owner from corresponding user attributes such as user id. The details of such constraints are not material for our purpose in this paper, hence we do not explicitly model them. The approach of $ABAC_\alpha$ [11] in this regard could be adapted to $ABAC_0$.

Actions are allowed operations in the system. These operations typically include create, read, update and delete. We use the terms actions and operations interchangeably. An action is applied to an object by a user. The term action is more commonly used in ABAC whereas operation is more common in the RBAC literature. An RBAC permission is defined to be an object, operation pair, which terminology we also use in this paper.

Actions are evaluated by authorization policy to enable access of a user to an object. Authorization policy is expressed as a propositional logic predicate for each action in the system, which takes as input a user and an object. Based on the values of the user and object attributes the authorization predicate for a given action returns true or false.

We formalize the above in the following definition, specifying sets, functions and authorization policy language.

Definition 1. *Core $ABAC_0$ is defined by the basic component sets, functions and authorization policy language given below.*

- *U and O represent finite sets of existing users and objects respectively.*
- *A represents a finite set of actions available on objects. Typically $A = \{create, read, update, delete\}$.*
- *$UATT$ and $OATT$ represent finite sets of user and object attribute functions respectively.*
- *For each att in $UATT \cup OATT$, $Scope(att)$ represents the attribute's scope, a finite set of atomic values.*

- $attType : UATT \cup OATT \rightarrow \{set, atomic\}$, *specifies attributes as set or atomic valued.*
- *Each attribute function maps elements in* U *and* O *to atomic or set values as follows.*

$$\forall uatt \in UATT.uatt : U \rightarrow \begin{cases} Scope(uatt) \; if \; attType(uatt) = atomic \\ 2^{Scope(uatt)} \quad if \; attType(uatt) = set \end{cases}$$

$$\forall oatt \in OATT.oatt : O \rightarrow \begin{cases} Scope(oatt) \; if \; attType(oatt) = atomic \\ 2^{Scope(oatt)} \quad if \; attType(oatt) = set \end{cases}$$

- *For each* $a \in A$, $Authorization_a(u : U, o : O)$ *is a propositional logic predicate, defined using the following language:*
  - $\varphi ::= \varphi \wedge \varphi \mid \varphi \vee \varphi \mid (\varphi) \mid \neg\varphi \mid \exists x \in set.\varphi \mid \forall x \in set.\varphi \mid set \; \triangle \; set \mid atomic \in set \mid atomic \; \nabla \; atomic$
  - $set ::= setuatt(u) \mid setoatt(o)$
  - $atomic ::= atomicuatt(u) \mid atomicoatt(o)$
  - $\triangle ::= \subset \mid = \mid \subseteq \mid \nsubseteq$
  - $\nabla ::= < \mid = \mid \leq$
  - $setuatt \in \{uatt \mid uatt \in UATT \wedge attType(uatt) = set\}$
  - $setoatt \in \{oatt \mid oatt \in OATT \wedge attType(oatt) = set\}$
  - $atomicoatt \in \{oatt \mid oatt \in OATT \wedge attType(oatt) = atomic\}$
  - $atomicuatt \in \{uatt \mid uatt \in UATT \wedge attType(uatt) = atomic\}$

Core $ABAC_0$ is a simplified version of $ABAC_\alpha$ [11], suitable for our purpose in this paper. In particular it eliminates subjects as being distinct from users as is in $ABAC_\alpha$, and simply treats them to be equivalent.

3 Multi-Tenant $ABAC_0$ Model

In this section we build upon $ABAC_0$ to formulate a multi-tenant attribute-based access control model enabling cross-tenant collaboration which we designate as MT-$ABAC_0$. The model structure is depicted in Figure 3, adding the tenant (T) entity in addition to the users and objects of core $ABAC_0$. *Tenants* are isolated operation domains leased by cloud service consumers.

Each user and each object is uniquely owned by a single tenant. For this purpose the model requires each user to have a system defined attribute *userOwner* which is a many-to-one atomic-valued function from users U to tenants T. Note that the arrowhead indicate the many side of the function while the absence of an arrowhead represents the one side. Likewise the model requires each object to have a system defined attribute *objOwner* which similarly is a many-to-one atomic-valued function from objects O to tenants T.

Further, each user attribute and each object attribute is also uniquely owned by a single tenant, depicted respectively by the many-to-one atomic-valued functions *uattOwner* and *oattOwner* in Figure 3. The crucial concept is that each tenant is responsible for assigning values to attributes that it owns. With isolated tenants, a user can have assigned values only for those attributes owned

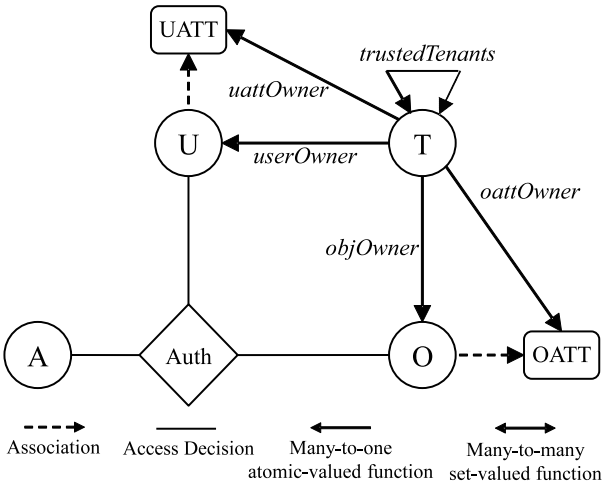

Fig. 3. Multi-Tenant $ABAC_0$ Model Structure.

by the user's owning tenant. We will see that, with appropriate trust relationship between tenants, users belonging to one tenant can be assigned values for attributes belonging to a different tenant. In our model for objects, we require that an object can have assigned values only for those attributes owned by the object's owning tenant. It is not possible for an object to be assigned values for attributes that belong to a tenant that does not own that object, regardless of tenant trust relationships. In summary cross-tenant attributes can be assigned to users under appropriate trust relationships but not to objects.

We define trust as a required attribute function *trustedTenants* mapping trustor tenant to trustee tenants which we refer to as tenant-trust. This is a many-to-many set-valued function. We use "$\trianglelefteq$" to represent the tenant-trust relation where $T_A \trianglelefteq T_B$ signifies that $T_B \in trustedTenants(T_A)$, i.e., T_B is trusted by T_A. In such cases we say T_A is the trustor tenant and T_B the trustee tenant. We have the following definition for tenant-trust.[1]

Definition 2. *If $T_A \trianglelefteq T_B$, Tenant T_B is authorized to assign values for T_B's user attributes to Tenant T_A's users. Tenant T_A controls tenant-trust existence while T_B controls cross-tenant attribute assignments.*

In general $\trianglelefteq$ is required to be a reflexive relation but is not required to be symmetric, anti-symmetric or transitive.

In light of the above definitions, we need to clarify the validity of attributes for users and objects. User attribute functions now become partial functions, because valid attribute values for a given user can only be assigned to certain user attributes. Specifically, a user u can be assigned a value for attribute *uatt* only if

[1] More generally different kinds of trust could be considered as discussed in Section 3.1.

$$uattOwner(uatt) = userOwner(u) \vee$$
$$uattOwner(uatt) \in trustedTenants(userOwner(u))$$

Similarly object attributes are also partial functions which are defined only for object attributes which are from the object's owner tenant. Specifically, an object o can be assigned a value for attribute $oatt$ only if

$$oattOwner(oatt) = objOwner(o)$$

In other words trust enables cross-tenant assignment of user attributes but does not impact object attributes.

Finally, each authorization predicate must verify the compatibility of user and object attribute ownership. For this reason, any user attribute $uatt$ or object attribute $oatt$ used in a action's authorization predicate with respect to a particular user u and object o, must satisfy the following condition.

$$uattOwner(uatt(u)) = oattOwner(oatt(o)) \vee$$
$$oattOwner(oatt(o)) \in trustedTenants(uattOwner(uatt(u)))$$

The above considerations lead to the following definition.

Definition 3. *Multi-tenant $ABAC_0$ is defined by the following enhancement and modifications to core $ABAC_0$.*

- *U, O, and A are defined as in core $ABAC_0$.*
- *T represents a finite set of existing tenants.*
- *$UATT$, $OATT$, $Scope$, and $attType$ are defined as in core $ABAC_0$.*
- *$userOwner : (u : U) \rightarrow T$, required attribute function mapping user u to owner tenant t.*
- *$objOwner : (o : O) \rightarrow T$, required attribute function mapping object o to owner tenant t.*
- *$MATT = \{uattOwner, oattOwner\}$, required meta-attribute functions.*
  - *$uattOwner : (uatt : UATT) \rightarrow T$, meta attribute function, mapping user attribute $uatt$ to attribute owner tenant t.*
  - *$oattOwner : (oatt : OATT) \rightarrow T$, meta attribute function, mapping object attribute $oatt$ to attribute owner tenant t.*
- *$trustedTenants : (t : T) \rightarrow 2^T$, required attribute function, mapping tenant t to powerset of trusted T, called tenant-trust, written as $\trianglelefteq$ where $t_1 \trianglelefteq t_2$ iff $t_2 \in trustedTenants(t_1)$ (i.e., trustor tenant t_1 trusts trustee tenant t_2). Trustee tenant t_2 can assign its attribute values $uatt_{t_2}$ to users u_{t_1} from trustor tenant t_1 where $t_2 \in trustedTenants(userOwner(u))$.*
- *Each attribute function $uatt \in UATT$ is modified to be a partial function.*
  $\forall uatt \in UATT.uatt : U \hookrightarrow \begin{cases} Scope(uatt) \; if \; attType(uatt) = atomic \\ 2^{Scope(uatt)} \quad if \; attType(uatt) = set \end{cases}$
  $uatt(u : U)$ is defined only if $(uattOwner(uatt) = userOwner(u)) \vee (uattOwner(uatt) \in trustedTenants(userOwner(u)))$.
- *Each attribute function $oatt \in OATT$ is modified to be a partial function.*
  $\forall oatt \in OATT.oatt : O \hookrightarrow \begin{cases} Scope(oatt) \; if \; attType(oatt) = atomic \\ 2^{Scope(oatt)} \quad if \; attType(oatt) = set \end{cases}$
  $OATT(o : O)$ is defined only if oattOwner(oatt) = objOwner(o).

– $\forall a \in A$, *Authorization*$_a(u : U, o : O)$ *is a propositional logic predicate (using language defined in ABAC*$_0$*), with the additional required condition that* $uattOwner(uatt(u)) = oattOwner(oatt(o)) \vee oattOwner(oatt(o)) \in trustedTenants(uattOwner(uatt(u)))$ *which must always be included in conjunction with all other requirements.*

3.1 Concept of Tenant Trust

In a tenant trust relation, in general there are two issues: (i) who controls trust relation's existence, and (ii) who has the authority to issue cross-tenant assignments. Together these characterize the trust type. In this paper, for simplicity, we adopted a specific definition of trust where trustee tenant is authorized to assign its attribute values to trustor tenant's user attributes which is analogous to the type-β tenant-trust of [24–26]. In this section, we briefly discuss trust types analogous to the type-α and type-γ tenant-trust types of [24–26].

In type-α trust, the trustor is responsible to establish the trust relationship with the trustee, as well as assigns the trustor's attributes to the trustee's users. We use $\trianglelefteq_\alpha$ to show this trust type where $T_A \trianglelefteq_\alpha T_B$ indicates that $T_B \in trustedTenants(T_A)$. With this notation, type-α tenant-trust is defined as follows.

Definition 4. *If* $T_A \trianglelefteq_\alpha T_B$*, Tenant* T_A *is authorized to assign values for* T_A*'s user attributes to Tenant* T_B*'s users. Tenant* T_A *controls tenant-trust existence and cross-tenant attribute assignments.*

In type-α trust, valid attribute values for given users are from owner tenants and trustor tenants. A user is assigned a value for an attribute *uatt* only if

$$uattOwner(uatt) = userOwner(u) \vee$$
$$userOwner(u) \in trustedTenants(uattOwner(uatt))$$

Each authorization predicate in type-α must satisfy following user and object attribute ownership condition.

$$uattOwner(uatt(u)) = oattOwner(oatt(o)) \vee$$
$$uattOwner(uatt(u)) \in trustedTenants(oattOwner(oatt(o)))$$

In type-γ trust, by trusting a tenant, trustor authorizes trustee to assign its attribute values to trustee tenant user attributes.We use $\trianglelefteq_\gamma$ to represent type-γ tenant-trust where $T_A \trianglelefteq_\gamma T_B$ signifies that $T_B \in trustedTenants(T_A)$. We define type-$\gamma$ trust as follows.

Definition 5. *If* $T_A \trianglelefteq_\gamma T_B$*, Tenant* T_B *is authorized to assign values for* T_A*'s user attributes to Tenant* T_B*'s users. Tenant* T_A *controls tenant-trust existence while* T_B *controls cross-tenant attribute assignments.*

Type-γ user attribute assignment and authorization predicate conditions are similar to above mentioned conditions in type-α. Type-γ differs from type-α, in which participating tenant has cross-tenant attribute assignment authority.

In relation to figure 1, when software development (SD) tenant trusts software testing (ST) tenant with type-β, it authorizes ST tenant to assign its attribute

values to software developers such as Alice to access resources in ST tenant. In type-α and Type-γ tenant-trust enables ST users such as John to access resources in SD tenant, where in type-α SD tenant assigns its attributes to John and in type-γ ST tenant assigns SD attribute values to its user John.

4 MT-ABAC$_0$ Model Covering MT-RBAC$_0$

In this section we first give a definition of multi-tenant RBAC (MT-RBAC$_0$) adapted from various slightly different but related models given in [24–26]. We then show how MT-RBAC$_0$ can be configured in MT-ABAC$_0$.

4.1 Multi-Tenant RBAC$_0$ Model

MT-RBAC$_0$ model element sets and relations are illustrated in Figure 4, showing the six components: tenants (T), users (U), roles (R), operations (OPS), objects (OBS), and permissions ($PRMS$). A *user* is an individual which is associated with a single tenant via UO relation. We recognize *role* as a job function associated with a single tenant while a tenant has multiple roles. *Objects* are tenant resources in the system (each object has a single owner tenant) which are coupled with *operations*. In RBAC, *permissions* are operation, object pairs indicating operations on objects.

MT-RBAC$_0$ model is defined in terms of users, roles, and objects owned by tenants. These ownership relations are many-to-one representing tenant ownership which is depicted as user-ownership (UO), role-ownership (RO), and object-ownership (OO) in figure 4.

As core to RBAC, user assignment (UA) and permission assignment (PA) relations enable assignment of users and permissions to roles. Tenant-trust (TT)

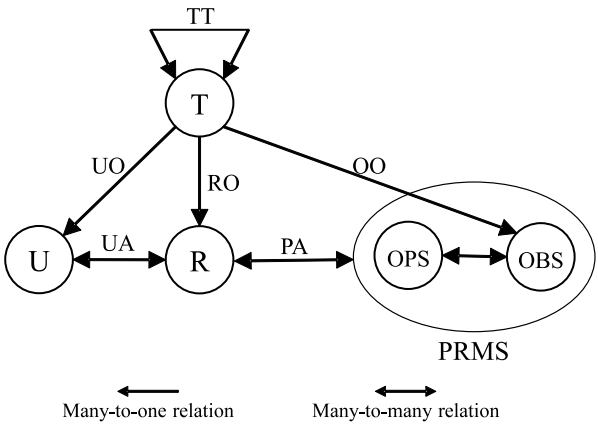

Fig. 4. Multi-Tenant RBAC$_0$ Model Structure.

identifies a many-to-many trust relation between tenants. Similar to MT-ABAC$_0$ we use $\trianglelefteq$ to show trust between two tenants such T_A and T_B as $T_A \trianglelefteq T_B$ means trustor tenant T_A trusts, trustee tenant T_B. With this specification, we define tenant-trust relation as follows.

Definition 6. *If $T_A \trianglelefteq T_B$, Tenant T_B is authorized to assign Tenant T_A's users to T_B's roles.*

In such trust relation, trusting a tenant enables trustee to assign trustor's users to its set of roles. This type of trust is intuitive in a sense that resource owners control access to their shared resources while user domains control their users' access by granted authority over trust relation continuation.

With existence of trust between tenants, user assignment is defined as many-to-many user relation mapping users to roles, if and only if users and roles owned by the same tenant or user owner tenant trusts object owner tenant. We express user assignment condition as $owner\_user(u) = owner\_role(r)$ (where *owner_user* returns owner tenant of user u and *owner_role* returns role r owner tenant) or $owner\_user(u) \trianglelefteq owner\_role(r)$. Permission assignment is a many-to-many relation which maps permissions to roles requiring both elements owned by the same tenant.

Each user is assigned to one or many roles within their resident tenants or trusted tenants. The function *assigned_user_roles* returns the roles assigned to a user. The permissions available to a user, are permissions assigned to roles (permissions available to a role are expressed by function *assigned_permissions*) that are available to a user which are given by function *authorized_user_permissions*. Function *authorized_user_permissions* designates set of permissions available to each user in the system. We formally define MT-RBAC$_0$ as follows.

Definition 7. *Multi-tenant RBAC$_0$.*

- *$TENANTS$, $USERS$, $ROLES$, OPS, and OBS (tenants, users, roles, operations, and objects respectively).*
- *$t \in TENANTS$, $u \in USERS$, $r \in ROLES$, $op \in OPS$, and $ob \in OBS$.*
- *$PRMS = OPS \times OBS$, the set of permissions.*[2]
- *$UO \subseteq USERS \times TENANTS$, a many-to-one user-to-tenant owner relation.*
- *$RO \subseteq ROLES \times TENANTS$, a many-to-one role-to-tenant owner relation.*
- *$OO \subseteq OBS \times TENANTS$, a many-to-one object-to-tenant owner relation.*
- *$owner\_user : (u : USERS) \rightarrow TENANTS$, the mapping of user u into its owner tenant. Formally: $owner\_user(u) = t$ where $(u, t) \in UO$.*
- *$owner\_role : (r : ROLE) \rightarrow TENANTS$, the mapping of role r into its owner tenant. Formally: $owner\_role(r) = t$ where $(r, t) \in RO$.*
- *$owner\_object : (ob : OBS) \rightarrow TENANTS$, the mapping of object ob into its owner tenant. Formally: $owner\_object(ob) = t$ where $(o, t) \in OO$.*

[2] This is slightly different from NIST standard model where $PRMS = 2^{(OPS \times OBS)}$, and more appropriate for our purpose.

- $TT \subseteq TENANTS \times TENANTS$, *is a many-to-many reflexive relation on* $TENANTS$ *called tenant trust relation, written as* $\trianglelefteq$ *where* $t_1 \trianglelefteq t_2$ *(trustor tenant* t_1 *trusts trustee tenant* t_2*) only if all users of* t_1 *can be assigned to roles of* t_2.
- $trustee\_tenants : (t : TENANTS) \rightarrow 2^{TENANTS}$, *the mapping of tenant* t *into a set of trusted tenants. Formally:* $trustee\_tenant(t) = \{t' \in TENANTS \mid t \trianglelefteq t'\}$.
- $UA \subseteq USERS \times ROLES$, *a many-to-many mapping user-to-role assignment relation requiring that* $(u, r) \in UA \Rightarrow owner\_user(u) = owner\_role(r) \vee owner\_user(u) \trianglelefteq owner\_role(r)$.
- $PA \subseteq PRMS \times ROLES$, *a many-to-many mapping permission-to-role assignment relation requiring that* $((op, ob), r) \in PA \Rightarrow owner\_object(ob) = owner\_role(r)$.
- $assigned\_roles : (op : OPS, ob : OBS) \rightarrow 2^{ROLES}$, *the mapping of object operation pair* (op, ob) *into a set of roles. Formally:* $assigned\_roles(op, ob) = \{r \in ROLES \mid ((op, ob), r) \in PA\}$.
- $assigned\_user\_roles : (u : USERS) \rightarrow 2^{ROLES}$, *the mapping of user* u *into a set of roles. Formally:* $assigned\_user\_roles(u) = \{r \in ROLES \mid (u, r) \in UA\}$.
- $assigned\_permissions : (r : ROLES) \rightarrow 2^{PRMS}$, *the mapping of role* r *into a set of permissions. Formally:* $assigned\_permissions(r) = \{p \in PRMS \mid (p, r) \in PA\}$.
- $authorized\_user\_permissions : (u : USER) \rightarrow 2^{PRMS}$, *the mapping of user* u *into a set of permissions.* $authorized\_user\_permissions(u) = \bigcup_{r \in assigned\_user\_roles(u)} assigned\_permissions(r)$.

4.2 Configuring MT-RBAC$_0$ to MT-ABAC$_0$

We show configuring MT-RBAC$_0$ in MT-ABAC$_0$ by adding role as an attribute function. Once roles become attributes, the consideration that roles are collections of permissions no longer applies since they are merely attribute values. Consequently, we must define appropriate object attributes and authorization predicates in MT-ABAC$_0$. To represent user assigned roles in MT-RBAC$_0$ ($assigned\_user\_roles$ function), we use a set-valued attribute function $userRole$. However users may be assigned roles owned by distinct tenants, for this purpose we identified user attributes as $userRole_j$ where j represents tenants.

In order to represent permission assignment, we define attribute function $objRole$ as a set-valued attribute function. Attribute $objRole$ captures roles related to each object in RBAC (permissions assigned to roles represented by $assigned\_roles$ function). In RBAC each object is owned by a tenant and coupled with a set of operations, for this reason we designate object attributes as $objRole_{i,k}$ where i is an operation in RBAC and k is owner tenant. The scope of both $userRole$ and $objRole$ attributes are the same as defined set of role names $ROLES$. We represent role ownership (RO) in RBAC by atomic-valued meta-attributes, $uattOwner$ and $oattOwner$ respectively mapping user and object

role attributes (*userRole* and *objRole*) to owner tenants. In the presence of roles attributes, authorization policy evaluates user and object respective role name attributes to be equal as well as user and object attributes ownership.

The summary of above is formalized as follows.

Definition 8. *A given MT-RBAC$_0$ instance is configured in MT-ABAC$_0$ as follows.*

- $U = USERS$, $O = OBS$, $A = OPS = \{a_1, ..., a_n\}$ *where* $n = |A|$, *and* $T = TENANTS = \{t_1, ..., t_m\}$ *where* $m = |T|$.
- $UATT = \{userRole_j \mid j = 1, ..., |T|\}$.
- $OATT = \{objRole_{i,k} \mid i = 1, ..., |A|, k = 1, ..., |T|\}$.
- $userOwner : (u : U) \rightarrow T$, *required attribute function, mapping user* u *to owner tenant* t. *Formally:* $userOwner(u) = owner\_user(u)$.
- $objOwner : (o : O) \rightarrow T$, *required attribute function, mapping object* o *to owner tenant* t. *Formally:* $objOwner(o) = owner\_object(o)$.
- $userRole_j : (u : U) \rightarrow 2^{ROLES}$ *where* $t_j \in T$, *attribute function, mapping user* u *to powerset of* $ROLES$. *Formally:* $userRole_j(u) = \{r \in ROLES \mid r \in assigned\_user\_roles(u) \wedge owner\_role(r) = t_j\}$.
- $objRole_{i,k} : (o : O) \rightarrow 2^{ROLES}$ *where* $a_i \in A$ *and* $t_k \in T$, *attribute function, mapping object* o *for operation* a_i *to powerset of* $ROLES$. *Formally:* $objRole_{i,k}(o) = \{r \in ROLES \mid r \in assigned\_roles(a_i, o) \wedge owner\_role(r) = t_k\}$.
- $MATT = \{uattOwner, oattOwner\}$.
  - $uattOwner : (userRole_j : UATT) \rightarrow T$, *meta attribute function, mapping user role attribute* $userRole_j$ *to attribute owner tenant* t_j. *Formally:* $uattOwner(userRole_j) = t_j$.
  - $oattOwner : (objRole_{i,k} : OATT) \rightarrow T$, *meta attribute function, mapping object role attribute* $objRole_{i,k}$ *for operation* a_i *to attribute owner tenant* t_k. *Formally:* $oattOwner(objRole_{i,k}) = t_k$.
- $trustedTenants : (t : T) \rightarrow 2^T$, *attribute function, mapping tenant* t *to powerset of trusted* T. *Formally:* $trustedTenants(t) = trustee\_tenants(t)$.
- $Authorization_i\ (u : U, o : O) = \bigvee_{k=1,...,|T|} [userRole_k(u) \cap objRole_{i,k}(o) \neq \emptyset \wedge (t_k = userOwner(u) \vee t_k \in trustedTenants(userOwner(u))]$.

5 Related Work

Several attribute-based access control models and systems have been proposed. In [9,10], ABAC and its functional components, implementation, and operation considerations are illustrated. This serves as an overview of components rather than considering modeling issues. Jin et al. [11] proposed ABAC$_\alpha$ model, designed to cover simple forms of DAC [21], MAC [19], and RBAC [6,20]. While this provides a realistic family of attribute-based models within single tenant environments, it does not consider collaboration and multi-tenancy issues. Smari et al. [22] investigated trust and privacy in collaborative management systems.

They extend attributes associated with objects and subjects to address trust and privacy issues. Although collaboration is considered, multi-tenancy has not been addressed.

Other approaches extending RBAC with combination of attributes and roles have been studied widely. Kuhn et al. [13] presented a spectrum of possible methods to combine RBAC and ABAC, specifically a policy-enhanced RBAC to accommodate attribute based features. However, the attributes are limited to user-centered attributes. In RABAC [12], authors integrate roles and attributes using a role centric approach. Parameterized role [3], object sensitive role [7], and attributed role [27] have also been proposed in this context.

Recent work on multi-tenancy collaboration such as CTTM [24] and OSAC-DT [25] extends RBAC to inherit its benefits towards collaboration. CTTM enables trust between tenants in a single cloud and OSAC-DT which is closely related to CTTM further extends it towards compatibility with OpenStack [2] platform. Tang [23] specifies a multi-tenant attribute based access control enabling cross-tenant access for subjects. Our model differs in structure and cross-tenant access where attribute value assignment enables such collaboration.

In order to benefit the RBAC capabilities in multiple organizations, prior extensions such as ROBAC [29] and GB-RBAC [15] have been proposed. ROBAC manages authorization in multiple organizations which is comparable to multi-tenancy, however organization collaboration is not considered in this context. In GB-RBAC collaboration is allowed among groups, yet it lacks the administration management since the administrator can not manage users in the groups. Role-based delegation [4,8,28] models have been proposed to permit collaboration, however chained delegation relations are not dynamic and flexible enough to be deployed in multi-tenant collaborative environments since trust relations in such collaborations are dynamic.

6 Conclusion

We presented a multi-tenant attribute-based access control model for resource sharing, where collaboration is enabled through cross-tenant attribute value assignments supported by the cloud service provider. In our proposed approach, we identified trust as a required attribute for tenants where trustee tenants are authorized to assign attribute values to trustor tenants' user attributes. In our approach, we eliminated attribute conflicts in presence of attribute assignments by isolating attributes to tenants. We believe our approach is applicable to other types of trust beyond presented trust types. A potential future work is to extend this model to address various types of trust. Another future research is extending our model to multi-cloud environments. Finally, our vision is to develop an implementation within current cloud platforms.

Acknowledgement. This research is supported by NSF Grant CNS-1111925 and CNS-1423481.

References

1. Amazon AWS. http://aws.amazon.com/es/ec2
2. OpenStack. http://www.openstack.org/
3. Abdallah, A.E., Khayat, E.J.: A formal model for parameterized role-based access control. In: Dimitrakos, T., Martinelli, F. (eds.) FAST 2005. IFIP, vol. 173, pp. 233–246. Springer, Heidelberg (2005)
4. Barka, E., Sandhu, R.: Framework for role-based delegation models. In: Proc. of Annual Conf. on Comp. Sec. Applications (ACSAC), pp. 168–176. IEEE (2000)
5. Coyne, E., Weil, T.R.: ABAC and RBAC: Scalable, flexible, and auditable access management. IT Professional **3**, 14–16 (2013)
6. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. TISSEC **4**(3), 224–274 (2001)
7. Fischer, J., Marino, D., Majumdar, R., Millstein, T.: Fine-grained access control with object-sensitive roles. In: Drossopoulou, S. (ed.) ECOOP 2009. LNCS, vol. 5653, pp. 173–194. Springer, Heidelberg (2009)
8. Freudenthal, E., Pesin, T., et al.: dRBAC: distributed role-based access control for dynamic coalition environments. In: Proc. of ICDCS, pp. 411–420. IEEE (2002)
9. Hu, V.C., Ferraiolo, D., et al.: Guide to attribute based access control (ABAC) definition and considerations. NIST Special Publication **800**, 162 (2014)
10. Hu, V.C., Kuhn, D.R., Ferraiolo, D.F.: Attribute-based access control. Computer **2**, 85–88 (2015)
11. Jin, X., Krishnan, R., Sandhu, R.S.: A unified attribute-based access control model covering DAC, MAC and RBAC. DBSec **12**, 41–55 (2012)
12. Jin, X., Sandhu, R., Krishnan, R.: RABAC: role-centric attribute-based access control. In: Kotenko, I., Skormin, V. (eds.) MMM-ACNS 2012. LNCS, vol. 7531, pp. 84–96. Springer, Heidelberg (2012)
13. Kuhn, D.R., Coyne, E.J., Weil, T.R.: Adding attributes to role-based access control. Computer **6**, 79–81 (2010)
14. Kurmus, A., Gupta, M., Pletka, R., Cachin, C., Haas, R.: A comparison of secure multi-tenancy architectures for filesystem storage clouds. In: Kon, F., Kermarrec, A.-M. (eds.) Middleware 2011. LNCS, vol. 7049, pp. 471–490. Springer, Heidelberg (2011)
15. Li, Q., Zhang, X., Xu, M., Wu, J.: Towards secure dynamic collaborations with group-based RBAC model. Computers & Security **28**(5), 260–275 (2009)
16. Mell, P., Grance, T.: The NIST definition of cloud computing (2011)
17. Pustchi, N., Krishnan, R., Sandhu, R.: Authorization federation in IaaS multi cloud. In: Proc. of Security in Cloud Computing, pp. 63–71. ACM (2015)
18. Sandhu, R.: The authorization leap from rights to attributes: maturation or chaos? In: Proc. of SACMAT, pp. 69–70. ACM (2012)
19. Sandhu, R.S.: Lattice-based access control models. Computer **26**(11), 9–19 (1993)
20. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer **29**(2), 38–47 (1996)
21. Sandhu, R.S., Samarati, P.: Access control: principle and practice. IEEE Communications Magazine **32**(9), 40–48 (1994)
22. Smari, W.W., Clemente, P., Lalande, J.-F.: An extended attribute based access control model with trust and privacy: Application to a collaborative crisis management system. Future Generation Computer Systems **31**, 147–168 (2014)
23. Tang, B.: Multi-Tenant Access Control for Cloud Services. PhD thesis, University of Texas at San Antonio (2014)

24. Tang, B., Sandhu, R.: Cross-tenant trust models in cloud computing. In: Proc. of Int. Conf. IRI, pp. 129–136. IEEE (2013)
25. Tang, B., Sandhu, R.: Extending openstack access control with domain trust. In: Au, M.H., Carminati, B., Kuo, C.-C.J. (eds.) NSS 2014. LNCS, vol. 8792, pp. 54–69. Springer, Heidelberg (2014)
26. Tang, B., Sandhu, R., Li, Q.: Multi-tenancy authorization models for collaborative cloud services. In: Proc. of CTS, pp. 132–138. IEEE (2013)
27. Yong, J., Bertino, E., Roberts, M.T.D.: Extended RBAC with role attributes. In: Proc. of PACIS, pages 457–469 (2006)
28. Zhang, X., Oh, S., Sandhu, R.: PBDM: a flexible delegation model in RBAC. In: Proc. of SACMAT, pp. 149–157. ACM (2003)
29. Zhang, Z., Zhang, X., Sandhu, R.: ROBAC: Scalable role and organization based access control models. In: Proc. of CollaborateCom, pp. 1–9. IEEE (2006)