MULTI-TENANT ACCESS CONTROL FOR CLOUD SERVICES

APPROVED BY SUPERVISING COMMITTEE:

Ravi Sandhu, Ph.D., Chair

Kay A. Robbins, Ph.D.

Gregory B. White, Ph.D.

Weining Zhang, Ph.D.

Jaehong Park, Ph.D.

Accepted:

Dean, Graduate School

DEDICATION

I would like to dedicate this dissertation to all my family and friends. A special feeling of gratitude to my loving parents for their words of encouragement and supports of great love. My grandparents have always been building themselves as examples for me to understand the greatness of life and future. My aunts, uncles and cousins have never left my side and are special.

I also dedicate this dissertation to my many friends who have supported me throughout the process. I will always appreciate all they have done, especially Tom and Mary Misuraca for helping me adapt to the life in San Antonio and go through hard times with their wisdom and care. Lijuan Dai, Yun Zhang, Fang Wang, Hui Shen, Yuan Xu, Jun Ye, Zhen Gao, Wenyuan Xiao, Jian Cui, Zhenxin Zhan, Li Xu, Weiliang Luo, Xin Jin, Xiaohuang Zhu and many more have shared with me their wonderful journey of pursuing further education and better life across the oceans and are important to my growth.

MULTI-TENANT ACCESS CONTROL FOR CLOUD SERVICES

by

BO TANG, M.S.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
August 2014

ACKNOWLEDGEMENTS

Sincere gratitude is hereby extended to the following individuals and organizations who help me accomplish this dissertation with their supervision, companionship and sponsorship.

Dr. Ravi Sandhu, for his wisdom, guidance and support of both my research and life throughout my doctoral studies. Dr. Qi Li, for his mentoring of my research skills and co-authorship of my first papers. This dissertation would not have been accomplished without their inspiring ideas, critical comments and constant encouragement.

The other committee members of this dissertation: Dr. Kay A. Robbins, Dr. Gregory B. White, Dr. Weining Zhang, Dr. Jaehone Park and Dr. Rajendra V. Boppana (in Proposal), for their insightful comments and the time they devoted to reading this dissertation.

My fellow Ph.D. students in the Institute for Cyber Security (ICS) lab: Yuan Cheng, Xin Jin, Dang Nguyen, Khalid Zaman Bijon, Yun Zhang, Tahmina Ahmed, Prosunjit Biswas, Navid Pustchi, Zhenxin Zhan, Li Xu, Qingji Zheng, Weiliang Luo and many more, for their companionship, sharing of ideas and passion.

Farhan Patwa, director of ICS, for his patient help on the OpenStack implementation and active connections with the OpenStack community.

Dolph Mathews, the Program Technical Lead (PTL) of Keystone, for his reviews and comments of the OSAC model and the domain trust blueprint.

This work is partially supported by grants from the National Science Foundation and AFOSR MURI program and the UT TRANSFORM program. Parts of Chapter 3 and Chapter 4 have appeared as [52,53,55]. Chapter 5 has been accepted and will appear in NSS 2014 as [54].

August 2014

MULTI-TENANT ACCESS CONTROL FOR CLOUD SERVICES

Bo Tang, Ph.D.
The University of Texas at San Antonio, 2014

Supervising Professor: Ravi Sandhu, Ph.D., Chair

Multi-tenancy is one of the key features of cloud computing. In order to protect data security and privacy for each customer (tenant), cloud service providers (CSPs) apply multi-tenant schemes to their shared services. Basically, a tenant, in its lifespan, owns a share of on-demand cloud resources such as users, virtual machine (VM) instances and storage containers. With the service-oriented architecture (SOA), all the services in a cloud need to support multi-tenancy and conform with a consolidated authorization model. We call such models as multi-tenant access control (MTAC) models which are compatible with the features of the cloud, namely, centralized facility, agility, homogeneity and out-sourcing trust. MTAC models should be able to address both intra-tenant and cross-tenant accesses. The former provides authorization schemes for single-tenant scenarios. The latter enables collaboration among tenants, which is an emerging trend of cloud evolution.

Multi-domain access control in traditional environments has been researched in various aspects such as role-based models, policy composition and decomposition, enforcement models and so on. However, the prior work is not directly applicable in the cloud environment or requires extra infrastructure for operation and administration. Furthermore, it is challenging for existing multi-domain models to encompass attribute-based access control (ABAC) which provides more expressiveness and flexibility especially meaningful in the cloud.

In this dissertation, we present a systematic research of MTAC models with a top-down approach. Our contributions are categorized into three layers: policy, enforcement and implementation (PEI). Starting from the policy (P) layer at the top, we propose a suite of MTAC models including role-based models, attribute-based models and cross-tenant trust models. The role-based models, MT-RBAC and MTAS, extend the traditional RBAC model to function in multi-tenant cloud environment and integrate two kinds of trust relations between tenants. Cross-tenant trust models provide a taxonomy of trust relations in terms of authorization for cross-tenant accesses. The trust models are also applicable to the attribute-based model, MT-ABAC, which similarly extends the ABAC$_\alpha$ model by means of cross-tenant trust. The P layer work builds a theoretical foundation and a framework of trust relations in cloud-based collaborative access control.

The enforcement (E) layer in the middle addresses the architecture of how the policies in the upper layer can be enforced to the implementations in the lower layer. Since the cloud has logically centralized infrastructure, we propose a novel Multi-Tenant Authorization as a Service (MTAaaS) to accommodate all the multi-tenant access control needs in a centralized service. The performance and scalability of this service is assured by the cloud. In this setting, the policies are stored in the central service along with the policy decision point (PDP). Each cloud service has a policy enforcement point (PEP) sending access requests to the PDP and enforcing responses from the PDP in a multi-tenant fashion. This architecture is prototyped using XACML implementation in cloud environment.

The implementation (I) layer at the bottom integrates the MTAC models into the real-world cloud system. We investigate OpenStack, one of the most popular open-source cloud systems and extend its identity service, Keystone, with a domain-trust module which enables multi-domain access control for OpenStack services. The domains in OpenStack are identical with tenants from our point of view. The results of experiments show minimum performance overhead with this newly introduced functionality.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: INTRODUCTION

The growing predominance of cloud computing impacts every aspect of the information technology (IT) industry [46]. It facilitates business agility and lowers costs for information systems by using virtualization techniques over shared infrastructures in an on-demand self-service manner. Due to the shared infrastructure, multi-tenancy is introduced as the key functioning model of cloud services [43] segregating customer data and work space for security and privacy purposes. Thus, the access control mechanisms supporting multi-tenant services are similar to those for distributed environments, but they are still different because of the following characteristics of typical cloud systems.

- **Centralized Facility.** Cloud services are usually presented as a pool of computing resources which are centralized in the CSP managed facility. The consumers (tenants) should be authorized full control of their temporarily "owned" resources including administration privileges. In other words, a tenant can create users within its scope and arbitrarily grant access to them. Hence, in this setting, pure decentralized models [40, 58] are not suitable.

- **Self-Service Agility.** The creation and deletion of tenants are not controlled by the CSP. A tenant may be created by the consumer for temporary use and deleted immediately afterwards. Thus, the authorization model, especially for cross-tenant accesses, has to be flexible enough to cope with such agility.

- **Homogeneity.** For the service operation of a cloud to meet a standard service level agreement (SLA), the CSP tend to build and maintain cloud systems with homogeneous architecture while only the customer configurations are different. Therefore, the access control model in different tenants should be identical with each other while the policies are customizable.

- **Out-Sourcing Trust.** Cloud users intrinsically out-source part of their IT infrastructures to CSPs so that trust relations between these two parties are already established. Collaborations among tenants also need similar trust relations, which can be developed through their common trust in the CSP.

The above listed features distinguish the cloud from other distributed systems in terms of multi-tenant authorization. It follows that access control models developed for previous distributed environments are not directly applicable to the cloud. Moreover, since multi-tenant collaborations are essential in the cloud [37], the multi-tenant access control (MTAC) models should address not only intra-tenant but also cross-tenant accesses. In particular, fine-grained secure resource sharing among tenants is not enabled in today's commercial clouds [34, 39].

1.1 Motivation

Before we discuss the multi-tenant access control problem, we need to clarify the definition of tenants. We see from the CSP's perspective that a tenant is a billable account owned by an organization, a department of an organization or even an individual. A tenant is not only a scope of operations but also an independent authority and an accessing entity. As a result, unlike domains in distributed environments, tenants in the cloud should be treated as entity components in the model rather than an attribute of users or objects. The formal definition of tenants is given in Chapter 3.

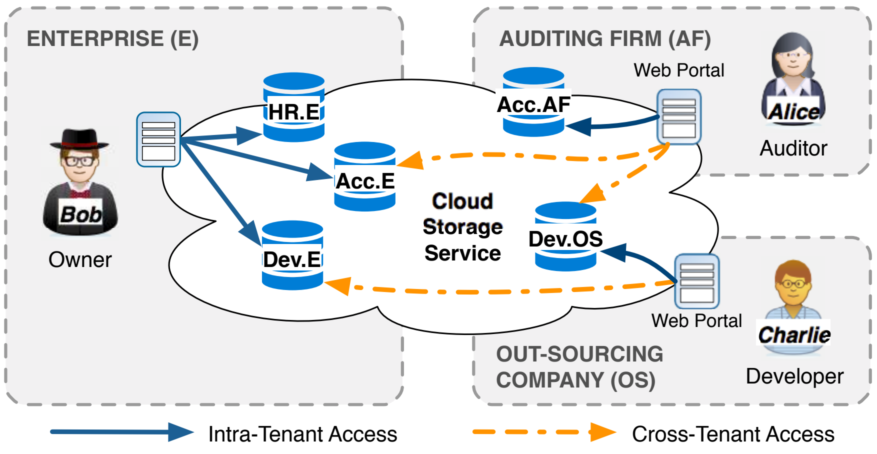


Figure 1.1: An out-sourcing case illustrating multi-tenant accesses

To better explain the problem, we use an out-sourcing example, as shown in Figure 1.1, in which the *Enterprise* (E), the *Out-Sourcing company* (OS), and the *Auditing Firm* (AF) are three collaborating parties sharing a common cloud storage service. E out-sources part of its application development work to OS and external auditing tasks to AF. The cloud storage service provides storage services for the development department of OS, the accounting department of AF, and three of E's departments, development, accounting, and HR, as segregated tenants. Let "." denote the affiliation relation between the tenant and its organization (also called issuer) so that, for example, $Dev.E$ represents the tenant of E's development department. The example cross-tenant accesses for collaborations are as follows.

C1. *Charlie* as a *developer* in OS has to access the source code stored in $Dev.E$ to perform his out-sourcing job;

C2. *Alice* as an *auditor* in AF requires read-only access to financial reports stored in $Acc.E$; and

C3. *Alice* needs read-only accesses to $Dev.E$ and $Dev.OS$ in order to audit the out-sourcing project.

For simplicity, in our examples we assume all the tenants are created on a single cloud service, bringing homogeneous architecture which is often the case in cloud environments [28]. But, we do not exclude the potential of heterogeneous collaborations among multiple cloud services or even among multiple service models: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [43]. As a common collaboration need suggests, a user may test and deploy the source code stored in $Dev.E$ directly on another tenant of a PaaS service. This function requires secure collaborative accesses between the two services. In fact, SaaS services are often hosted on PaaS clouds in which the SaaS services are regarded as tenants. A similar situation holds between PaaS and IaaS. Thus, we treat accesses among multiple tenants, clouds, or even service models equivalently in the abstraction level as multi-tenant accesses upon which access control mechanisms should be enforced.

Currently, CSPs use Single Sign-On (SSO) techniques to achieve authentication and simple authorization in federated cloud environments, but fine-grained authorization is typically not supported. NASA has integrated role-based access control (RBAC) into Nebula [42], a private cloud system. While traditional RBAC enables fine-grained access control mechanisms in clouds, it lacks the ability to manage collaborations. IBM [26] and Microsoft [25] 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 [40] and dRBAC [29], 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. Consequently, in order to enable secure multi-tenant collaborations in the cloud, we need a general fine-grained access control model for this purpose.

To achieve collaborations among cloud services, Calero et al [22] proposed a multi-tenancy authorization system 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.

1.2 Problem Statement

Contemporary cloud systems extend the role-based access control (RBAC) model to address multi-tenant authorization. This exercise matches the centralized facility and homogeneity characteristics of the cloud, since RBAC is intrinsically a centralized model initially designed for a single organization. However, in terms of self-service agility and out-sourcing trust, these models limit the flexibility of the cloud and typically disallow cross-tenant accesses or support these only minimally. The permissions in RBAC are presented as operations against objects which are not identifiable in the policy since they are temporary. As a result, the authorization policies can only specify the relation between roles and operations as role-permission assignments so that a role can only be associated with a set of operations in a tenant. Moreover, current cloud systems apply limited control on the cross-tenant accesses. Some of them treat cross-tenant accesses more or less identical as intra-tenant accesses with no extra control which is obviously needed. Hence, the current access control solutions in cloud systems do not meet all the requirements and characteristics of the cloud environment identified here.

In order to cover the major multi-tenant authorization requirements and cloud characteristics, we feel it necessary to build a generic formal model since the previous models are not directly applicable to the cloud. This model is a mixture of centralized and decentralized models. On the one hand, all the tenants in all the services of a cloud have to comply with a uniform model which is established by the CSP and enforced by each service. On the other hand, each tenant has full control of the cloud computing resources within its scope. The model should be feasible in all the three cloud service models: IaaS, PaaS and SaaS. Since IaaS is the foundation of the other two models and less complicated in use cases, it is a reasonable environment to begin experimenting with. Also, there are requirements for finer-grained access control. It is challenging to allow administrators to specify authorization policies based upon objects before they are created. This can be achieved using attribute-based access control (ABAC) approaches.

In order to develop the model described above, we still have some challenges in front of us. Here follows the challenges we identify.

- **Common Vocabulary.** During the process of specifying cross-tenant authorization policies, a tenant needs to understand the roles or other attributes in the context of another tenant and vice versa. As a result, establishing a common vocabulary between tenants is essential.

- **Policy Conflicts.** The policies specified in the user tenant and the object tenant of an access may conflict with each other. The policy decision point (PDP) should resolve the conflict using a predefined algorithm.

- **Centralized or Decentralized PDP.** The policy enforcement points (PEPs) are distributed in each service in a multi-tenant fashion. The PDP can be hosted either in each cloud service or in a centralized authorization service. The former needs communication between services to retrieve information needed in the policy decision process but is more efficient since the PDP is closer to the cloud resources. The major drawback of the latter is the performance issue since all the services will communicate with the authorization service for each access so that the implementation have to be scalable in order to meet the high performance requirements.

- **Administration Model.** The administration of authorization policies is the concern of most access control models. In multi-tenant cloud environments, the administration model should allow the CSP to conduct daily operations but not intervene with tenant policies.

- **Constraints.** The tenants may also need to specify constraint policies to better control multi-tenant accesses. For example, separation of duty (SoD) in a single tenant can be covered by local tenant policies but in a multi-tenant scenario, the same problem may require CSP level policies or remote policies to specify the constraint.

1.3 Scope and Assumptions

This research is conducted based on the following assumptions.

Standardized APIs. The CSP should have a set of standardized APIs and other necessary facilities, in order to functionally enable cross-tenant accesses. Our research mainly focuses on the access control of these APIs.

Authenticated Users. All the users requesting accesses are assumed to have been properly authenticated.

These two assumptions are more or less necessary for any treatment of cross-tenant trust. The rest of the assumptions below could be relaxed or removed but are convenient for an initial investigation of cross-tenant trust models.

One Cloud Service. For simplicity, we assume cross-tenant accesses are between tenants of a single cloud service. We believe our models are extensible beyond a single cloud but multi-cloud considerations are outside the scope of this dissertation.

Two Tenant Trust. For simplicity we only consider trust relations between two tenants. More generally, there may exist trust relations for more than two tenants forming a community, a coalition, or a federation [18].

Unidirectional Trust Relations. Each trust relation is unidirectional (like follow in Twitter) as opposed to bidirectional (like friend in Facebook).

Unilateral Trust Relations. 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, modify, and revoke a trust relation. In general, it seems unreasonable for the trustee to unilaterally assert a trust relationship. However, it may be reasonable for a trustee to agree before a trustor can assert trust. Also, it may be reasonable for a trustee to unilaterally revoke trust with respect to a trustor. Treatment of these cases is outside the scope of this dissertation.

1.4 Thesis

The central thesis of this dissertation is as follows.

The problem of multi-tenant access control in the cloud can be partially solved by integrating various types of unidirectional and unilateral trust relations between tenants into role-based and attribute-based access control models.

1.5 Summary of Contributions

The contributions in this dissertation are summarized into three aspects.

- **Policy.** The multi-tenant authorization system in [22] was formalized as a role-based MTAS model and extended with finer-grained trust relations between tenants. Furthermore, we developed another model, the Multi-Tenant Role-Based Access Control (MT-RBAC) model, similar to MTAS but with a different type of trust relation. We proposed a taxonomy of the Cross-Tenant Trust Models (CTTM) in multi-tenant access control. Finally, the Multi-Tenant Attribute-Based Access Control (MT-ABAC) model was developed to integrate more flexibility and granularity into the policy.

- **Enforcement.** We proposed a centralized architecture called Multi-Tenant Authorization

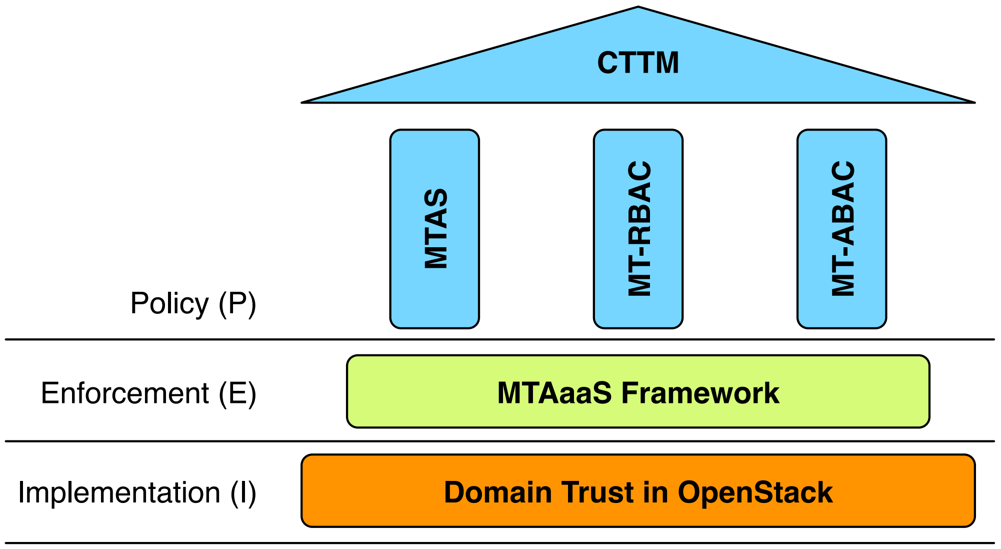


Figure 1.2: MTAC Framework Structure

as a Service (MTAaaS) to enforce the policy layer models in the cloud. The specification of policies were composed in standard eXtensible Access Control Markup Language (XACML). Also, we built a prototype system based on SUN's XACML implementation and conducted experiments to validate the feasibility of the models and the service architecture.

- **Implementation.** We presented the OpenStack Access Control (OSAC) model in our perspective and integrated a novel domain trust relation in the implementation of Keystone, the identity service of OpenStack. In the identity service API v3 [7], domain is equivalent to tenant in our models. The PDP was implemented in both centralized and decentralized ways. The experiments were conducted over the decentralized architecture because it is consistent with the Havana release [6] of OpenStack. The results show that the extended domain trust implementation introduces minimum performance overhead and thereby maintains the scalability of Keystone.

Following the PEI framework for application-centric security [47], we organize our research in three layers: Policy (P), Enforcement (E) and Implementation (I). The entire structure of our work towards multi-tenant access control is illustrated in Figure 1.2. The accomplished works are mainly in P layer and E layer. In future work, we plan to keep developing the P layer models and refining the MTAaaS framework in the E layer as well. The models shown in P layer in Figure 1.2 are not an exhaustive list. Additional multi-tenant access control models are planed to be investigated in future work. Then, our eventual goal is to bring an I layer MTAaaS platform to cloud users and developers leveraging OpenStack.

1.6 Organization of the Dissertation

Chapter 2 gives a literature review of the related works including multi-domain, delegation and Grid solutions. Chapter 3 presents the policy layer works including MTAS, MT-RBAC, CTTM and MT-ABAC. Chapter 4 describes the MTAaaS architecture enforcing multi-tenant access control models in the cloud. Chapter 5 gives the formal definition of OSAC model, the domain trust extension and experiment results of the implementation. Chapter 6 concludes the dissertation and discusses the future work.

Chapter 2: RELATED WORK

Multi-tenant access control shares some basic problems with multi-domain secure interoperation approaches in traditional distributed environments. Some solutions have already been proposed in multi-domain circumstances. We summarize these approaches into four categories: centralized approaches, decentralized approaches, attribute-based approaches and enforcement models.

2.1 Centralized Approaches

Role-based access control (RBAC) [27] enabling fine-grained access control does not encompass the overall context associated with any collaborative activity [56]. Many RBAC extensions have been proposed to address multi-domain access control in the context of roles. A typical way to solve the context mismatch during multi-domain authorization process is establishing a common centralized authority for all the domains collaborating with each other.

Shafig et al [49] propose a set of policy composition mechanisms using role mapping and integer program (IP) to solve heterogeneity issues in secure multi-domain information sharing. This approach introduces graph-based specification model for RBAC and clarifies the fundamental problems in cross-domain authorization through roles. Zhang et al [59] propose a scalable role and organization based access control model (ROBAC) to scale up RBAC for modeling security policies spanning multiple organizations. ROBAC is designed specifically for organizations with homogeneous structure in a hierarchical way. It aggregates roles and assets with organizations and asset types respectively significantly shrinking the size of role-based access control policies. Another RBAC extension, the group-based RBAC model (GB-RBAC) [41], is designed to foster dynamic collaborations in distributed environments such as the Grid. With the group entity introduced, GB-RBAC allows a user to be assigned a permission either through system-level roles or group-level roles. The former is consistent with the user-role assignment in RBAC while the latter is extended to accommodate groups of users. In this way, the administration is effectively separated into two levels, the group-level and system-level, so that some administrative privileges can be delegated to group-level administrators. In the above approaches, a centralized authority is required to specify or mediate cross-domain policies. However, in cloud environment, the only existing centralized authority is the Cloud Service Provider (CSP) who should maintain the generic policies for all the tenants rather than for specific ones since the tenants are temporary and self-service oriented. Thus, these approaches are not directly applicable in the cloud.

2.2 Decentralized Approaches

In order to facilitate access control in distributed systems, another line of work advocates decentralized authorities extending RBAC. Role-based trust management framework (RT) [40] provides localized authority over roles, delegation in role definition, linked roles and parameterized roles using a few simple credential forms. It is theoretically expressive and adaptive to distributed systems. Similarly, distributed RBAC (dRBAC) [29] utilizes PKI credentials to define trust domains, roles to define controlled activities, and role delegation across domains to represent permissions to these activities. It is designed for highly dynamic coalition environments with multiple organizations or entities that are unwilling to rely on a third party to administer trust relationships. Permission-Based Delegation Model (PBDM) [58] extends RBAC with flexible role and permission level delegations. In PBDM, a security administrator specifies the permissions that a user (delegator) has authority to delegate to others (delegatees), then the delegator creates one or more temporary delegation roles and assigns delegatees to particular roles. This mechanism provides clear separation of security administration and delegation. It allows transitive delegations giving control of permissions to user for decentralized management. Barka et al [17] propose a framework for role-based delegation models giving a taxonomy of delegation models. This framework clarifies the basic characteristics of delegation and its usage in RBAC.

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 (users) change, the entire graph of authorization will change unexpectedly.Thus, the delegation approaches also lack support of the agility in the cloud.

2.3 Attribute-Based Approaches

The benefit of Attribute Based Access Control (ABAC) is prominent in multi-domain access control [33]. In RBAC and many other previous models, access to an object has to be individually granted to locally identified subjects so that the external subject's identity has to be pre-provisioned in the object's domain prior to the access request. However, ABAC avoids the need for explicit authorizations to be directly assigned to specific subjects but to subjects with certain attributes. Thus, with ABAC the authorization policies for both intra-domain and cross-domain accesses are unifiable as long as the data sharing agreements and infrastructures are established.

The concept of ABAC has existed for many years. It is anticipated to be the next generation of dominant access control models. From our perspective, ABAC is more generic than RBAC since the role is an attribute of users as well as permissions. Some of the attributes can be conveyed as roles but a lot more cannot be. For example, the context attributes such as the accessing location and time are not straightforward to be described by roles. At the same time, the flexibility of attributes further complicates the access control problems. As a result, a commonly accepted unifying ABAC model does not yet exist.

Although ABAC models are still under development, its core entities and basic concepts have already been specified. The earlier approach is to add attributes to the existing RBAC model. Al-Kahtani et al [13] add attribute feature into the user-role assignment process enabling dynamic and automatic assignments. The composite model is capable to express Mandatory Access Control (MAC). Kuhn et al's work [38] summaries and compares the possible ways to integrate attributes with RBAC (RBAC-A). In general, there are three approaches namely dynamic roles, attribute-centric and role-centric to achieve the RBAC-A models. Recently, Xin et al [36] propose a unified ABAC$_\alpha$ model covering DAC, MAC and RBAC. The ABAC$_\alpha$ model clearly describes the relations among users, subjects and objects, as well as their attributes using its policy configuration language. There are three configuration points in ABAC$_\alpha$ for specifying attribute constraint policies and authorization policies. The authorization decision is computed based on these policies. In a more recent work of these authors [35], the ABAC$_\beta$ model is proposed to cover more traditional models including a considerable number of RBAC extensions.

2.4 Enforcement and Implementation

Besides the policy layer work in the literature, there is significant related work in enforcing and implementing access control mechanisms in multi-domain environments, especially in the cloud. Federated identity and authorization services are included in the architecture of distributed systems to facilitate collaborative access control. Federated identity [19, 24] enables authenticating strangers by sharing identity information among federated parties. The federation relation is intrinsically an equal trust relation which is meaningful in some use cases but cumbersome in supporting the variety of collaborations. For example, the trust relation between E and OS in Figure 1.1 is apparently not an equal trust since the ultimate owner of the resources to be accessed is E. 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, such as VOMS [14], PERMIS [23] and CAS [44], aim to secure resource sharing among virtual organizations (VOs) 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 due to the existence of centralized facility and homogeneous architecture.

To enforce access control in web environments, Yuan et al [57] propose an ABAC enforcement model. The architecture contains a centralized policy decision point (PDP) and distributed policy enforcement points (PEP). The attributes and authorization policies are specified by distributed attribute authorities and policy authorities. This enforcement model is designed for web services but also feasible in other SOA systems.

As a critical part of the evolving cloud technology, authorization mechanisms for multi-tenant collaborations are emerging in both academia and industry. Multi-tenancy in the cloud features homogeneity, centralized facility, and agility comparing to the traditional multi-domain environments. Utilizing the centralized facility feature in clouds [15, 22, 30, 51] it is possible to develop authorization services with scalable policy management modules and PDPs in a central location. Moreover, to better clarify the trust relationships in authorization domains for cloud computing, Perez et al [45] recently propose 29 different trust models and 8 fine-grain models in terms of conditions, subjects and targets applying to a universal trust operator. They also give a centralized trust manager architecture for IaaS clouds. This enforcement model is consistent with this dissertation.

Chapter 3: MULTI-TENANT ACCESS CONTROL (MTAC) MODELS

In this Chapter, we propose a suite of four MTAC models including two role-based models, one trust model and one attribute-based model, as shown in the Policy layer of Figure 1.2. All the role-based and attribute-based models use various types of tenant trust relations, as described in the Cross-Tenant Trust Model (CTTM), to bridge authorization between tenants. CTTM defines three types of tenant trust relations as Type-α, Type-β and Type-γ trust. The role-based models, Multi-Tenant Authorization System (MTAS) and Multi-Tenant Role-Based Access Control (MT-RBAC), use Type-β and Type-γ trust respectively. The attribute-based model, Multi-Tenant Attribute-Based Access Control (MT-ABAC), is compatible with all the three types of trust relations.

3.1 MTAS

In this section we formalize the multi-tenancy authorization system informally described in [22]. We call the resulting model as the MTAS model for ease of reference and continuity. We also develope an administration model for MTAS (called the AMTAS model). Further, we propose two feasible enhancements to the trust model of MTAS.

3.1.1 Formalization

Towards the goal of a general model of multi-tenant role-based access control in the cloud, we start by abstracting the MTAS system [22] into a formalized model, as shown in Figure 3.1. There are four entity components: *tenants* (T),[1] *users* (U), *permissions* (P) and *roles* (R). In addition to classic RBAC$_2$, the role hierarchy model [27], the *tenant* component is introduced to express authorization in multi-tenant environments, while other components need to be modified accordingly. In particular the traditional RBAC entities of *users*, *permissions* and *roles* have *tenant* attributes so that they can be identified uniquely in a multi-tenant cloud environment. This is depicted by the

[1]The *tenant* entity is generated from the concept of "issuer" in [22] for consistency and clarity.

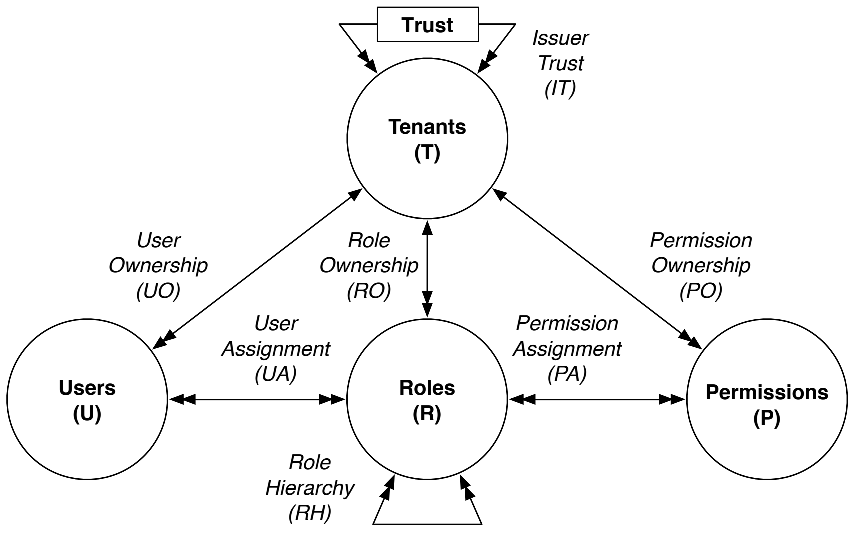


Figure 3.1: An abstracted model of the MTAS system.

UO, RO and PO relations in Figure 3.1. UO, RO and PO are many-to-one relations from U, R and P respectively to T.

TENANTS. A *tenant* represents an organization, a department of an organization or even an individual who uses the cloud services. From the CSP point of view, each *tenant* is an independent customer.The data and operations of a *tenant* are isolated from each other. For example, in the out-sourcing case described in Figure 1.1, E as an organization owns three tenants: $Dev.E$, $Acc.E$ and $HR.E$. They are used for different departments in E but the CSP treat them uniformly as cloud storage service customers.

USERS. A *user* is an identity for an individual. It is authenticated as a federated ID [19] which is universally unique for all the tenants in the community. Every user has an owner attribute indicating the tenant who provides the identity. The identity is also usable by other tenants.

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

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

SESSIONS.[2] 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 can 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 tenant.

Crucially, an additional tenant trust relation ($TT \subseteq T \times T$, also written as "$\lesssim$") establishes tenant to tenant trust as will be described and formalized in detail later in this section. For $\forall t_r, t_e, t_f \in T$, TT relation is reflexive

$$t_r \lesssim t_r \tag{3.1}$$

but not transitive

$$t_r \lesssim t_e \wedge t_e \lesssim t_f \nRightarrow t_r \lesssim t_f \tag{3.2}$$

and it is neither symmetric

$$t_r \lesssim t_e \nRightarrow t_e \lesssim t_r \tag{3.3}$$

nor anti-symmetric

$$t_r \lesssim t_e \wedge t_e \lesssim t_r \nRightarrow t_r = t_e. \tag{3.4}$$

For $t_r \lesssim t_e$, we call t_r the trustor and t_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 permission 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

[2]The session component was not discussed in [22], 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.

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 below.

Definition 1. *Let A and B denote two tenants. By establishing a tenant trust relation (TT) with B ($A \precsim B$), A exposes its entire role hierarchy 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 Figure 1.1, Bob, representing the resource owner $Dev.E$, 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 $Dev.E$ for the out-sourcing job, *(edit, Dev.E, /src/)* is associated to the role $role(Dev.E, dev)$. In order to achieve this cross-tenant access, with the presence of $Dev.OS \precsim Dev.E$ relation, Bob can assign $role(Dev.E, dev)$ to be a junior role of an appropriate developer role in $Dev.OS$, say $role(Dev.OS, dev)$. In this way, the users associated to $role(Dev.OS, dev)$ are able to edit the files under the */src/* directory in $Dev.E$.

The trust model solves the two key problems in collaborative role-based access control: decentralized authority and semantic mismatch. Since the collaborators are independent, the tenants (decentralized authorities) desire to maintain 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 while the trustee assigns permissions of its data to the trustor. In this way, both sides contribute to cross-tenant assignments and the accesses are under mutual control.

The semantic mismatch issue refers to the fact that the definitions of roles vary in different domains so that no proper assignment could be made by a single authority without additional communication with each other. In the trust model of MTAS, this issue is mitigated, because the authorization settings, i.e. the role hierarchy and the role members, of the trustor are exposed to the trustee upon the creation of the tenant trust relation. Consider the out-sourcing case. With the presence of $Dev.OS \lesssim Dev.E$, $Dev.E$'s administrator Bob may examine the members of $Dev.OS$'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, T and S (users, roles, permissions, tenants and sessions respectively);*
- *$UO \subseteq U \times T$, a many-to-one relation mapping each user to its owning tenant; correspondingly, $userOwner(u : U) \to T$, a derived function mapping a user to its owner where $userOwner(u) \in \{t \in T | (u, t) \in UO\}$;*
- *$RO \subseteq R \times T$, a many-to-one relation mapping each role to its owning tenant; correspondingly, $roleOwner(r : R) \to T$, a derived function mapping a role to its owner where $roleOwner(r) \in \{t \in T | (r, t) \in RO\}$;*
- *$PO \subseteq P \times I$, a many-to-one relation mapping each permission to its owning tenant; correspondingly, $permOwner(p : P) \to T$, a derived function mapping a permission to its owner where $permOwner(p) \in \{t \in T | (p, t) \in PO\}$;*
- *$TT \subseteq T \times T$, a reflexive relation on T called tenant trust relation, also written as $\lesssim$;*
- *$canUse(r : R) \to 2^T$, a derived function mapping a role to a set of tenants who can use the particular role. Formally, $canUse(r) = \{t \in T | roleOwner(r) \lesssim t\}$;*
- *$UA \subseteq U \times R$, a many-to-many user-to-role assignment relation;*
- *$PA \subseteq P \times R$, a many-to-many permission-to-role assignment relation requiring $(p, r) \in PA$ only if $permOwner(p) \in canUse(r)$;*
- *$RH \subseteq R \times R$ is a partial order on R called role hierarchy or role dominance relation, also written as $\geq$, requiring $r \geq r_1$, only if $roleOwner(r_1) \in canUse(r)$;*

- $user(s : S) \rightarrow U$, *a function mapping each session to a single user which is constant within the life-time of the session; and*

- $roles(s : S) \rightarrow 2^R$, *a function mapping each session to a subset of roles,* $roles(s) \subseteq \{r|(\exists r' \geq r)[(user(s), r') \in UA \wedge userOwner(user(s)) \in canUse(r)]\}$, *which can change within* s, *and* s *has the permissions* $\bigcup_{r \in roles(s)}\{p|(\exists r'' \leq r)[(p, r'') \in PA]\}$.

Note that since we are formalizing an extension of pure RBAC model [27], the user permission assignment described in [22] is ignored in the formalization.

Role activation mechanisms determine the executable permissions inherited by a session. Because a role may inherit permissions from its junior roles in the role hierarchy, when a role is activated in a session, its inherited roles may be either automatically activated (implicit activation) or require explicit activation. Theoretically the former scenario is transformable to the latter by recursively executing explicit activation for the junior roles. The choice between the two approaches is left as an implementation issue in the NIST RBAC model [27]. In the RBAC96 model implicit activation is specified [48]. In MTAS we choose to specify explicit activation in the $roles(s)$ component. In a session, only the permissions of the explicitly activated roles are executable to the user.

Since every user identity is available to all the tenants, UA assignments bear no constraints on tenants. Thus, the UA assignments are always issued by the role owners as discussed in the administration of the MTAS model in Section 3.1.2.

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 tenants who can use r to make authorization assignments. The returned tenants 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 tenants.

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.

3.1.2 Administrative MTAS (AMTAS) Model

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

Definition 3. *The Administrative MTAS (AMTAS) model 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-tenant 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-tenant access, the resource requesting tenant controls the trust relations which decides whether or not to allow cross-tenant 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-tenant authorization because if either is revoked or altered, the corresponding collaborative accesses will be denied.

Table 3.1: Administrative Functions in AMTAS

Function	**Condition**	**Update**
Administrative functions available to cloud administrators:		
AddTenant(t)	$t \notin T$	$T' = T \cup \{t\}$

RemoveTenant(t)	$t \in T$	**forall** $t_e \in T$ **do**
		  RevokeTrust(t, t_e)
		  RevokeTrust(t_e, t)
		forall $userOwner(u) \equiv t$ **do**
		  RemoveUser(t, u)
		forall $roleOwner(r) \equiv t$ **do**
		  RemoveRole(t, r)
		forall $permOwner(p) \equiv i$ **do**
		  RemovePerm(t, p)
		$T' = T \setminus \{t\}$

Administrative functions available to tenant *t*:		
AddUser(t, u)	$userOwner(u) \equiv t \wedge u \notin U$	$U' = U \cup \{u\}$
RemoveUser(t, u)	$userOwner(u) \equiv t \wedge u \in U$	**forall** $\{r : R \mid (u, r) \in UA\}$ **do** RevokeUser(t, u, r) $U' = U \setminus \{u\}$
AddRole(t, r)	$t = roleOwner(r) \wedge r \notin R$	$R' = R \cup \{r\}$

Continued on next page ≫

Table 3.1: Administrative Functions in AMTAS (Contd.)

Function	Condition	Update
RemoveRole(t, r)	$t = roleOwner(r) \wedge r \in R$	**forall** $\{u : U \mid (u, r) \in UA\}$ **do** RevokeUser(t, u, r) **forall** $\{p : P \mid (p, r) \in PA\}$ **do** RevokePerm(t, p, r) **forall** $\{r_{asc} : R \mid (r_{asc}, r) \in RH\}$ **do** RevokeRH(t, r_{asc}, r) **forall** $\{r_{desc} : R \mid (r, r_{desc}) \in RH\}$ **do** RevokeRH(t, r, r_{desc}) $R' = R \setminus \{r\}$
AddPerm(t, p)	$permOwner(p) \equiv t \wedge p \notin P$	$P' = P \cup \{p\}$
RemovePerm(t, p)	$permOwner(r) \equiv t \wedge p \in P$	**forall** $\{r : R \mid (p, r) \in PA\}$ **do** RevokePerm(t, p, r) $P' = P \setminus \{p\}$
AssignUser(t, u, r)	$t = roleOwner(r) \wedge u \in U$	$UA' = UA \cup \{(u, r)\}$
RevokeUser(t, u, r)	$t = roleOwner(r) \wedge u \in U \wedge (u, r) \in UA$	$UA' = UA \setminus \{(u, r)\}$
AssignPerm(t, p, r)	$t = permOwner(p) \wedge t \in canUse(r)$	$PA' = PA \cup \{(p, r)\}$
RevokePerm(t, p, r)	$t = permOwner(p) \wedge t \in canUse(r) \wedge (p, r) \in PA$	$PA' = PA \setminus \{(p, r)\}$
AssignRH(t, r_{asc}, r)	$t = roleOwner(r) \wedge t \in canUse(r_{asc}) \wedge \neg(r_{asc} \gg r) \wedge \neg(r \geq r_{asc})$ †	$\geq' = \geq \cup \{r_2, r_3 : R \mid r_2 \geq r_{asc} \wedge r \geq r_3 \wedge roleOwner(r_3) \in canUse(r_2) \bullet (r_2, r_3)\}$ ‡

Continued on next page ≫

Table 3.1: Administrative Functions in AMTAS (Contd.)

Function	**Condition**	**Update**
RevokeRH(t, r_{asc}, r)	$t = roleOwner(r) \wedge t \in canUse(r_{asc}) \wedge r_{asc} \gg r$	$\geq' = (\gg \setminus \{(r_{asc}, r)\})^{*}$ [§]
AssignTrust(t, t_e)	$t_e \in T$	$\precsim' = \precsim \cup \{(t, t_e)\}$
RevokeTrust(t, t_e)	$t_e \in T \wedge t \precsim t_e \wedge t \neq t_e$ [¶]	$\precsim' = \precsim \setminus \{(t, t_e)\}$ [♭]

[†] 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.

[¶] A tenant cannot refuse to trust itself. Otherwise, improper revocation of assignments may occur.

[♭] By revoking the trust relation, the canUse() function of $ttenant$'s roles automatically updates accordingly as well as PA and RH.

□

Table 3.1 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 tenants [3]. Along with the removal of a tenant, its correlated trust relations, users, roles and permissions should also be removed. Even though the users are globally available, the removal of their owning tenants will result in removal of the users as well since the identity of the users depends on their owners. The removal of users will result in revocation of correlated UA. As cross-tenant UA is allowed, some assignments authorized by other tenants may also be removed. The same situation happens when a tenant is trying to remove one of its roles. In this way, cross-tenant authorization assignments are controlled

[3]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.

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-tenant assignments (PA and RH) specified by the trustee are automatically removed or not are 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 since according to Definition 2 the authorization assignments will not function without the proper trust relation when the corresponding cross-tenant accesses are being checked.

3.1.3 Enhanced Trust Models

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

Trustor-Centric Public Role (TCPR)

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

- $\mathcal{P}_T(t:T) \rightarrow 2^R$*, a function mapping an tenant to a set of its public roles which are the only roles that* i *expose to its trustees; and*
- $canUse(r:R) \rightarrow 2^T$ *is modified to* $canUse(r) = \{t\} \cup \{t_1 \in t | t \lesssim t_1 \wedge r \in \mathcal{P}_T(t)\}$*, where* $t = roleOwner(r)$*.*

By introducing $\mathcal{P}_T(t)$, the exposure surface of the t's roles in TCPR is much smaller than that in MTAS trust model. Accordingly, only if $r \in \mathcal{P}_T(t)$, then r can be used by t's trustees. Otherwise, it can only be used internally by t .

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

Relation-Centric Public Role (RCPR)

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

- $\mathcal{P}_R(tt : TT) \rightarrow 2^R$*, a function mapping a tenant trust relation to a set of the trustor's public roles; and*
- $canUse(r : R) \rightarrow 2^T$ *is modified to* $canUse(r) = \{t\} \cup \{t_1 \in I | t \lesssim t_1 \land r \in \mathcal{P}_R(t \lesssim t_1)\}$*, where* $t = roleOwner(r)$*.*

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.

The tenant trust relation (Type-β) in MTAS allows the trustee to give access to the trustor. It is applicable in some use cases that the trustor is willing to expose its users and roles to the trustee. However, in some other use cases, the trustor treats its users as sensitive information and does not want to disclose even to the trustees. Thereby, we propose another type (Type-γ) of tenant trust relation allowing the trustee to take access from the trustor. In this way, only roles and permission need to be expose to the trustees. Similar to MTAS, MT-RBAC, as proposed in the following section, facilitates multi-tenant access control using Type-γ trust.

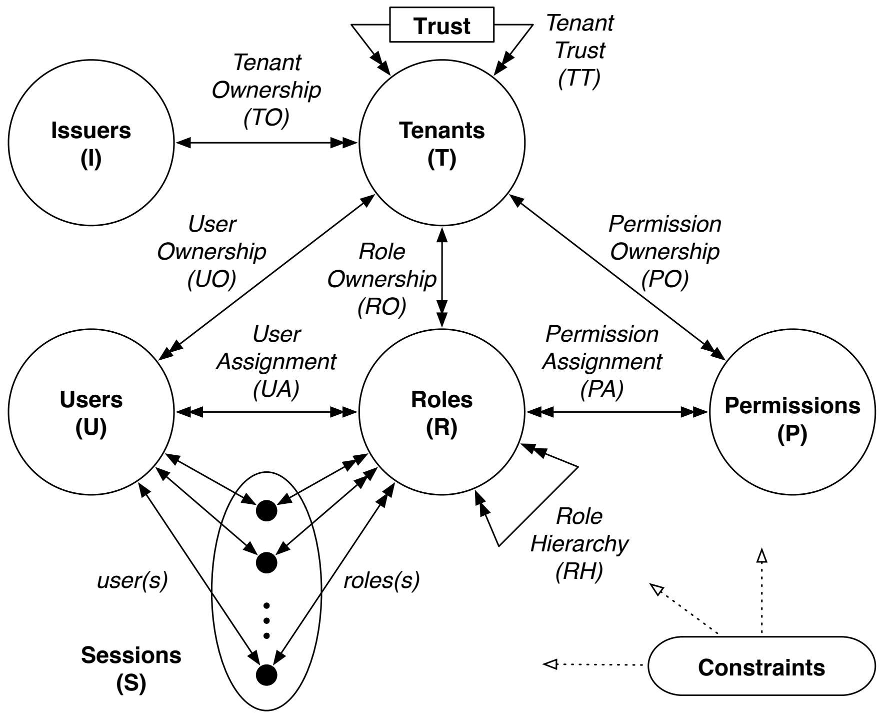


Figure 3.2: MT-RBAC Model

3.2 MT-RBAC

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

3.2.1 Overview

MT-RBAC models, as shown in Figure 3.2, have six entity components: *issuers* (I), *tenants* (T), *users* (U), *permissions* (P), *roles* (R) and *sessions* (S). The traditional RBAC [27] entities of users, permissions and roles now have a tenant attribute so that they can be uniquely identified as depicted by the user-ownership (UO), permission-ownership (PO) and role-ownership (RO) relations respectively in Figure 3.2. All three relations are many-to-one relations from users, permissions and roles respectively to their owner tenants. Further, another many-to-one relation representing tenant-ownership (TO) exists between tenants and issuers.

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

TENANTS. A tenant is an exclusive virtual partition of a cloud service leased from a cloud service provider. An issuer may own multiple tenants while a tenant belongs to a single issuer. Let "." denote the tenant-issuer relation. For example, $Dev.OS$ represents the tenant Dev of OS.[4]

USERS. A user is an identifier for an individual person associated with a single tenant. An individual can act as different users in different tenants. Let "@" denote the user-tenant relation. For example, $Alice@Acc.AF$ and $Alice@Acc.E$ are two different users in tenants $Acc.AF$ and $Acc.E$ respectively, even if they belong to a single person, Alice.[5]

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

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

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

Crucially, in order to address collaborations among tenants, MT-RBAC introduces a role-based trust relation, Tenant Trust (TT), as illustrated in Figure 3.2.

TT is reflexive but not transitive, symmetric or anti-symmetric. In MT-RBAC, a trust relation

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

[5]Other user models are possible. For instance, the users of a same person can be combined into one universal ID using federated identity [18, 19, 32]. However, since this mechanism is not fully supported in contemporary clouds, MT-RBAC models do not require a universal ID for an individual. The particular user model chosen does not materially impact the essentials of MT-RBAC. For completeness we do need a concrete user model.

is always established by the truster's issuer. It enables the trustee's issuer to add trustee's users to truster's roles. This can be done directly on a per user basis by assigning one of B's (trustee's) users to one of A's (truster's) roles via UA. Alternately it can be done indirectly by assigning one of B's roles, say r_1, to be senior to one of A's roles, say r_2, via RH. Thereby all members of r_1 become members of r_2 and A's permissions associated with r_2 are granted to the B's users in r_1. We emphasize that trust is established at the granularity of tenants. For example, if AF asserts $Acc.AF \trianglelefteq Acc.E$, there is no trust from $Acc.AF$ to $HR.E$ or $Dev.E$. The formalization of TT and its effects are described in Definition 6 as follows.

Definition 6. *The tenant trust relation, also written as "$\trianglelefteq$". By asserting $A \trianglelefteq B$, A's issuer exposes A's roles to B's issuer so that B's issuer can and can only make the following two kinds of assignments:*

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

Different MT-RBAC models vary in the granularity of the trust relations, specifically in the truster's role exposure. The left part of Figure 3.3 shows the role hierarchy of $Dev.E$ in the outsourcing example. Since MT-RBAC$_0$ does not enforce any constraint on the trust relation, the entire $Dev.E$ role structure is exposed to all $Dev.E$'s trustees (more precisely to their issuers). In MT-RBAC$_1$, suppose the employee role $emp\#Dev.E$ is a private role while the others are public. The private role is never exposed to other tenants. Conversely, the public roles are exposed to all the trustees' issuers. Crucially, during cross-tenant accesses, the permissions associated with the private role cannot be inherited directly or even indirectly through the public roles. In MT-RBAC$_2$, public role sets are customized for different trustees. Suppose the accountant role $acc\#Dev.E$ and the manager role $mgr\#Dev.E$ have to be exposed in the collaboration with $Acc.AF$ while $mgr\#Dev.E$ and the developer role $dev\#Dev.E$ need to be exposed to $Dev.OS$ for the outsourcing project. Accordingly, as illustrated in Figure 3.3, there are two different public role sets to these two trustees respectively. Note that even if $mgr\#Dev.E$ is public in both collaborations,

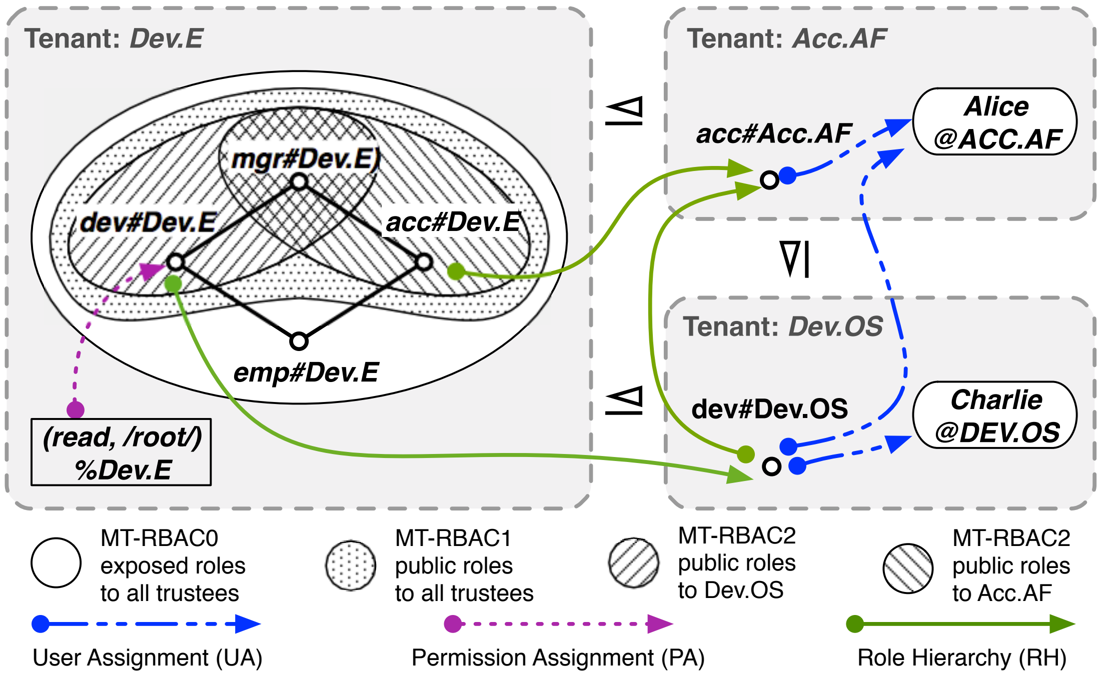


Figure 3.3: Example multi-tenant assignments in the out-sourcing case. The differences among the MT-RBAC models are also illustrated in terms of E's exposed roles.

for either one, it only inherits permissions from the junior roles in the corresponding public role set. For example, since $acc\#Dev.E$ is a private role to $Dev.OS$ and a public role to $Acc.AF$, its permissions are only inherited to $Acc.AF$ but not to $Dev.OS$.

3.2.2 Base Model—MT-RBAC$_0$

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

Definition 7. *The Base Model MT-RBAC$_0$ has the following components.*

- T, U, R, P *and* S *are finite sets of tenants, users, roles, permissions and sessions respectively;*
- $userOwner(u : U) \rightarrow T$*, a function mapping a tenant-specific user to its owning tenant;*
- $roleOwner(r : R) \rightarrow T$*, a function mapping a tenant-specific role to its owning tenant;*
- $permOwner(p : P) \rightarrow T$*, a function mapping a tenant-specific permission to its owning tenant;*

- $TT \subseteq T \times T$ *is a many-to-many reflexive relation on* T*, also written as "*$\trianglelefteq$*";*

- $canUse(r : R) \rightarrow 2^T$*, a function mapping a role to a set of tenants who can use the role; formally,* $canUse(r) = \{t \in T | roleOwner(r) \trianglelefteq t\}$*;*

- $UA \subseteq U \times R$*, a many-to-many user-to-role assignment relation requiring* $(u, r) \in UA$ *only if* $userOwner(u) \in canUse(r)$*;*

- $PA \subseteq P \times R$*, a many-to-many permission-to-role assignment relation requiring* $(p, r) \in PA$ *only if* $permOwner(p) = roleOwner(r)$*;*

- $RH \subseteq R \times R$ *is a partial order on* R *called role hierarchy or role dominance relation, also written as "*$\geq$*", requiring* $r_2 \geq r_1$ *only if* $roleOwner(r_2) \in canUse(r_1)$*;*

- $user(s : S) \rightarrow U$*, a function mapping each session to a single user which is constant within the life-time of the session; and*

- $roles(s : S) \rightarrow 2^R$*, a function mapping each session to a subset of roles,* $roles(s) \subseteq \{r | (\exists r' \geq r)[(user(s), r') \in UA \wedge userOwner(user(s)) \in canUse(r)]\}$*, which can change with time, and* s *has the permissions* $\bigcup_{r \in roles(s)} \{p | (\exists r'' \leq r)[(p, r'') \in PA]\}$.

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

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

Role activation mechanisms determine the executable permissions in a session. Because a role may inherit permissions from its junior roles in the role hierarchy, when a role is activated in a session, its inherited roles may be either automatically activated (implicit activation) or require explicit activation. The choice between the two approaches is left as an implementation issue in the NIST RBAC model [27]. In the RBAC96 model implicit activation is specified [48]. In MT-RBAC, we choose to specify explicit activation in the $roles(s)$ component. In a session, only the permissions of the explicitly activated roles are available. An alternate forumulation of MT-RBAC with implicit activation can also be developed.

The revocation of a TT relation $t_r \trianglelefteq t_e$ can be asserted by t_r's issuer. This operation will automatically eliminate the trustee t_e from $canUse(r)$ for each of $\{r \in R | (r, t_r) \in RO\}$. Moreover, as the formal description of UA and RH in Definition 7 suggests, all the relevant cross-tenant assignments (i.e., UA and RH) issued by t_e's issuer will be revoked automatically, as well as the active roles in the sessions of t_e's users. In this way the permissions in t_r are not able to be inherited to t_e after revocation.[6] If the removed trust relation is subsequently restored, the trustee's issuer would have to redefine and reissue all the cross-tenant assignments from scratch.

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

3.2.3 Trustee Independent Public Role—MT-RBAC$_1$

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

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

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

Definition 8. *MT-RBAC$_1$ inherits all the components from MT-RBAC$_0$ as described in Definition 7, while the following modifications are applied.*

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

Note that the truster's permissions associated with the public roles can be inherited externally to the trustees, but those associated with the private roles can only be inherited internally within the truster. For example, in Figure 3.3, since the employee role $emp\#Dev.E$ is a private role, its permissions should never be used in cross-tenant accesses. Further, if $\mathcal{P}_{TI}(t)$ consists of the entire role set of t, then MT-RBAC$_1$ is identical to MT-RBAC$_0$. Therefore, MT-RBAC$_1$ is a more general model than the base model.

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

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

3.2.4 Trustee Dependent Public Role—MT-RBAC$_2$

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

Definition 9. *MT-RBAC$_2$ inherits all the components from MT-RBAC$_0$ as described in Definition 7, while the following modifications are applied.*

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

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

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

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

3.2.5 Administrative MT-RBAC (AMT-RBAC) model

The administration of tenant trust relations and authorization assignments is critical in MT-RBAC. Since TT is embedded in the cross-tenant assignments (i.e., UA and RH) as described in Definition 7, the management of tenant trust relations also controls cross-tenant accesses.

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

Definition 10. *The Administrative MT-RBAC (AMT-RBAC) model requires both issuers of the collaborating tenants, i_A the resource owner A's issuer and i_B the requester B's issuer, to manage the tenant trust relations and the cross-tenant authorization assignments separately as follows.*

- *i_A is responsible for managing the tenant trust relation of $A \trianglelefteq B$; and*
- *i_B is responsible for managing the cross-tenant assignments (i.e., UA and RH) to A's roles, according to Definition 7.*

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

In Table 3.2, we give the formal specification of the exact administration functions of AMT-RBAC for a single issuer i along with the corresponding preconditions and updates to MT-RBAC policies.

Table 3.2: Administration functions available to tenant *t* in AMT-RBAC

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

† The notation "$\gg$" represents an immediate inheritance relation. For example, $r_{asc} \gg r_{desc}$ means that r_{asc} is a parent of r_{desc}.

‡ This condition avoids the creation of role cycles which is discussed in Section 3.5.2.

§ The notation "$*$" represents recursive updates for the entire RH assignments. Implied RH relations are preserved after revocation.

¶ The revocation of a trust relation automatically triggers updates in the $canUse()$ function of all t's roles and then corresponding UA and RH accordingly.

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

3.3 CTTM

In a social context, trust has several connotations. Definitions of trust typically refer to a situation characterized by the following aspects. One party (trustor) is willing to rely on the actions of another party (trustee) with respect to the future. In addition, the trustor (voluntarily or forcedly) abandons control over the actions performed by the trustee [8]. This definition of trust is also applicable in the virtual world, including cloud computing. For example, cloud consumers trust cloud providers to manage their data while cloud providers trust cloud consumers to use their computing resources responsibly. These two trust relations are both established by a service level agreement (SLA) which regulates the responsibilities of each party.

3.3.1 Motivation

The trust relation between two tenants of a cloud service provider is analogous to the trust relation between a car rental company, say AVIS, and a customer organization, say UTSA. Let $AVIS$ and $UTSA$ represent two tenants in a Platform as a Service (PaaS) [43] for AVIS and UTSA respectively. The PaaS is in charge of hosting applications for its tenants. Figure 3.4 illustrates an example of cross-tenant access needs between these two tenants.

Assume AVIS and UTSA have an agreement about discounted car rental price from AVIS exclusively for UTSA students. The agreement itself is an established trust relation created outside of the PaaS. In traditional practices, AVIS may give away coupons on UTSA campus or send

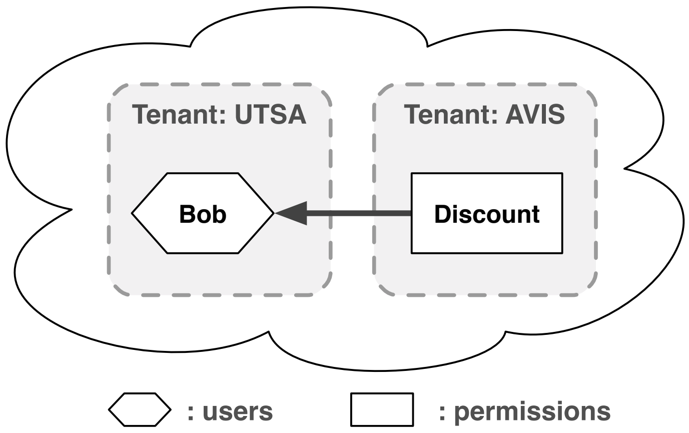


Figure 3.4: A car rental example of cross-tenant access.

coupon code to UTSA mailing lists. These approaches provide little control to the distribution of coupons and their use. Thus, controlling access of the discount privilege in the cyber domain is critical in the overall trust relationship.

In this example, the user information of UTSA is stored in the cloud, more specifically in the PaaS, where the discount privilege of AVIS can also be accessed. Thus, the access control mechanisms of cross-tenant accesses from $UTSA$ users to $AVIS$ permissions are provided by the PaaS. For instance, Bob as a student user in $UTSA$ wants to access the discount permission in $AVIS$, as shown in Figure 3.4. In order to securely enable this cross-tenant access, the PaaS should support an appropriate trust model regulating who builds the trust and who executes the trust.

3.3.2 On-Demand Self-Service in the Cloud

On-demand self-service is one of the essential characteristics of the cloud [43]. CSPs provide centralized facilities of computing resources which are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and re-assigned according to consumer demand. A consumer can unilaterally provision the computing resources as needed automatically without human interaction with the CSP. Moreover, the tenants and users can only be treated as temporary entities since a user can easily create a cloud user account, get a tenant in a cloud service for some tasks and release the tenant when the job is done. The user account may also be removed after usage.

In cross-tenant accesses, the self-service feature requires agility in the trust model. The trustor and the trustee may be created on-demand so that the trust relation between them should be established and destroyed easily. Additionally, a trust negotiation process for a bilateral trust may not be suitable in this environment. Instead, unilateral trust relations asserted by the trustor better match the on-demand self-service feature of the cloud.

Cross-Tenant Trust Model (CTTM) consists of different types of unilateral trust relations which reflect different needs in access control between two tenants, the trustor and the trustee. In this section, we first present an analysis of the tenant trust (TT) relations and discuss their types and usage. Then, we give a formalization of CTTM and its role-based extension (RB-CTTM). Furthermore, in order to argue the feasibility of the cross-tenant trust models in the cloud, we propose a multi-tenant authorization as a service (MTAaaS) platform to facilitate the enforcement.

3.3.3 Tenant Trust Relations

Before we discuss the formalization of CTTM, we first give an analysis of tenant trust (TT) relations which is the crucial part of our cross-tenant access control models. TT (also written as "$\trianglelefteq$") is a binary relation from the trustor to the trustee. Let "$\equiv$" denote the same tenant relation. For example, "$A \equiv B$" means that A and B are the same tenant. Let T be the set of all tenants. For $\forall A, B, C \in T$, a TT relation is reflexive

$$A \trianglelefteq A \tag{3.5}$$

but not transitive

$$A \trianglelefteq B \wedge B \trianglelefteq C \nRightarrow A \trianglelefteq C \tag{3.6}$$

and it is neither symmetric

$$A \trianglelefteq B \nRightarrow B \trianglelefteq A \tag{3.7}$$

nor antisymmetric

$$A \trianglelefteq B \land B \trianglelefteq A \nRightarrow A \equiv B. \tag{3.8}$$

Statement (3.5) requires that a tenant always trusts itself since intra-tenant accesses are not influenced by the trust relations. In order to control the propagation of trust relations and cross-tenant accesses enabled by the trust relations, Statement (3.6) requires that a trust relation can only be directly defined by the trustor but is never inferred from indirect combination of other trust relations. Statement (3.7) and (3.8) basically require that a trust relation is unidirectional and independent in each direction. A single tenant can be the trustor in one trust relation and the trustee in another. Together these statements characterize the building and basic characteristics of cross-tenant trust.

Turning to the usage of TT, we identify four potential types of trust relations to enable and control cross-tenant accesses.

- Type-α: trustor can give access to trustee.
- Type-β: trustee can give access to trustor.
- Type-γ: trustee can take access from trustor.
- Type-δ: trustor can take access from trustee.

The terms of "giving" and "taking" accesses distinguish the authorities of issuing cross-tenant assignments. Sticking with the car rental example, $AVIS$ giving access to $UTSA$ is equivalent to $AVIS$ assigning $AVIS$'s permissions to $UTSA$'s users. Conversely, $UTSA$ taking access from $AVIS$ means $UTSA$ can make the same assignment.

Type-α trust (also written as "$\trianglelefteq_\alpha$") is most intuitive since it is closest to familiar real world trust relations. For example, by establishing $AVIS \trianglelefteq_\alpha UTSA$, $AVIS$ can obtain user information in $UTSA$ and assign cross-tenant accesses from $UTSA$'s users to $AVIS$'s permissions. In this type of trust relation, the trustor ($AVIS$) holds the authority of assigning its own permissions to the trustee's users and requires visibility to the trustee's ($UTSA$'s) user information. Nevertheless, user information is also considered as sensitive information for $UTSA$ and $UTSA$ may wish to limit its exposure. Note that the trust is unilaterally asserted by the trustor $AVIS$ which enables visibility into $UTSA$'s user information without any involvement of $UTSA$. In such cases, Type-α trust is not suitable.

Type-β trust (also written as "$\trianglelefteq_\beta$") alters the direction of the trust relation in Type-α trust so that the trustor ($UTSA$) can control the exposure of its user information which is necessary for the trustee ($AVIS$) to make cross-tenant authorization assignments. In the car rental example, by establishing $UTSA \trianglelefteq_\beta AVIS$, $UTSA$ explicitly exposes its user information to $AVIS$ so that $AVIS$ can assign its permissions to $UTSA$'s users based on $UTSA$'s user information. In this way, access to the discount permission is controlled by both the trustor ($UTSA$) and the trustee ($AVIS$) together. No single party can unilaterally authorize a cross-tenant access.

In Type-γ trust (also written as "$\trianglelefteq_\gamma$"), the cross-tenant access is also controlled by both the trustor and the trustee together. But it is very different than Type-β trust. By establishing the trust relation, the trustor delegates the control of cross-tenant authorization assignments to the trustee. Thus, in order to maintain the control of cross-tenant accesses, the trustor doesn't issue cross-tenant assignments but just appropriately manage the trust relations which is required for the assignments to take effect. For instance, by establishing $AVIS \trianglelefteq_\gamma UTSA$, $AVIS$ delegates $UTSA$ to assign cross-tenant access from $UTSA$'s users to $AVIS$'s permissions. Because $UTSA$ is more familiar with the organization of its user information, $UTSA$ is more knowledgable to assign its users to $AVIS$'s permissions such as discounts. For instance $UTSA$ can determine which users within UTSA get the discount, e.g., full-time students but not part-time students.

Type-δ trust relation does not provide meaningful use cases since the trustor holds all the control of the cross-tenant assignments of the trustee's permissions. For example, $UTSA$, or any other tenants in the cloud service, can unilaterally trust $AVIS$ and assign $AVIS$'s permission to its own users. In this kind of situation, cross-tenant accesses cannot be controlled by the permission owners. Thus, Type-δ trust relation has little practical usage in cross-tenant access control, and we will ignore it henceforth.

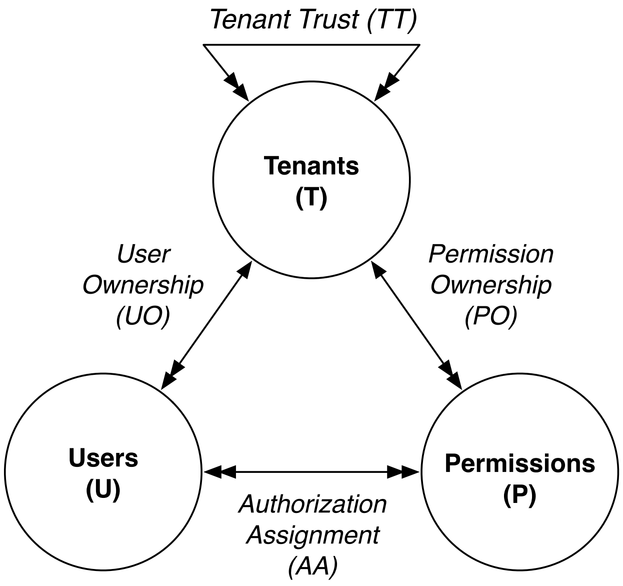


Figure 3.5: Cross-Tenant Trust Model

3.3.4 Formalized Model

In the formalization of CTTM, as shown in Figure 3.5, there are three entity components: *users* (U), *permissions* (P) and *tenants* (T). Both the U and P components exist in most of the formalized access control models since they are critical in expressing an access. A novel component T is introduced to express accesses in multi-tenant environments in which the other components should fit. In particular a user in U and a permission in P are owned respectively by a tenant in T so that they can be identified uniquely in a multi-tenant access in cloud environments. This is depicted by the UO and PO relations in Figure 3.5. PO is a many-to-one relation from P to T.

UO may be a many-to-one relation or a many-to-many relation from U to T depending on implementation. In a many-to-many case, a user may be associated with multiple tenants out of which if the permission owner trusts one or more, then the access can be granted to the user. Here, conflict of permissions[7] may happen during the policy evaluation process because various trust relations are invoked for one user. This problem is out of the scope of this dissertation. For simplicity, we choose to define UO as a many-to-one relation in CTTM formalization.

[7]Conflict may arise if negative permissions are allowed in the access control policy.

TENANTS. A tenant is an exclusive virtual partition of a cloud service leased from a CSP [52]. In practice a tenant is usually mapped to a project, a department, or an organization. Cloud user activities and resource accesses are defined within the domain of a tenant. For example, $UTSA$ is a tenant created for UTSA as an organization customer of the PaaS service so that UTSA can manage its users and resources in the domain of $UTSA$ tenant[8].

USERS. A user is an identifier for an individual or a process in a tenant. Each user has a single owner tenant while a tenant has multiple users. A user is formed by a username and a tenant and is written in the format of "username@tenant". Note that an individual or a process may possess multiple users in different tenants. These users are treated independently. In the car rental example, $Bob@UTSA$ is a user acting for Bob in $UTSA$.

PERMISSIONS. A permission is a specification of a privilege in a tenant. It is formed by a permission name and a tenant and is written in the format of "permission_name%tenant". Each permission has a single owner tenant while a tenant has multiple permissions. For example, $discount\%AVIS$ represents the discount permission in $AVIS$.

The formal definition of CTTM follows.

Definition 11. *The cross-tenant trust model (CTTM) has the following components.*

- T*,* U*, and* P *are finite sets of tenants, users, and permissions respectively;*
- $UO \subseteq U \times T$*, is a many-to-one relation mapping each user to its owner tenant; correspondingly,* $userOwner(u : U) \to T$*, is a function mapping a user to its owner tenant where* $userOwner(u) = t$ *iff* $(u, t) \in UO$*;*
- $PO \subseteq P \times T$*, is a many-to-one relation mapping each permission to its owner tenant; correspondingly,* $permOwner(p : P) \to T$*, is a function mapping a permission to its owner tenant where* $permOwner(p) = t$ *iff* $(p, t) \in PO$*;*

[8]In a more general treatment we would identify the cloud service explicitly in a two part name such as $UTSA.CloudService$.

- *$TT \subseteq T \times T$ is a many-to-many tenant trust relation on T, also written as "$\trianglelefteq$"; depending on the system TT can be one of Type-α, Type-β or Type-γ;*

- *$AA \subseteq U \times P$, a many-to-many user-to-permission assignment relation, also written as "$\leftarrow$", requiring that $u \leftarrow p$ only if*
  $permOwner(p) \equiv userOwner(u) \vee$
  $permOwner(p) \trianglelefteq_{\alpha} userOwner(u) \vee$
  $userOwner(u) \trianglelefteq_{\beta} permOwner(p) \vee$
  $permOwner(p) \trianglelefteq_{\gamma} userOwner(u)$,
  where only one of the $\trianglelefteq$ requirements can apply depending on the nature of TT.[9]

In Definition 11, AA represents a set of multi-tenant assignments, including cross-tenant and intra-tenant assignments. AA should comply with the conditions specified in the definition. Intra-tenant assignments are always prohibited, since $permOwner(p) \equiv userOwner(u)$ is always true and moreover TT is reflexive for each type of trust. For cross-tenant assignments, the permission owner should be the trustor either in a Type-α or a Type-γ trust relation, or the trustee in a Type-β trust relation, while the user owner is on the other end of the trust relations. For example, in order to enable the cross-tenant assignment "$Bob@UTSA \leftarrow discount\%AVIS$", the appropriate one of the following three trust relations should exist: $AVIS \trianglelefteq_{\alpha} UTSA$, $UTSA \trianglelefteq_{\beta} AVIS$, or $AVIS \trianglelefteq_{\gamma} UTSA$, depending on the nature of TT.

The revocation of cross-tenant authorization assignments in CTTM can be achieved in two ways. One way is revoking the assignment by the assignment issuer (executor of the trust relation) which is the trustor in Type-α and Type-β trust or the trustee in Type-γ trust. Since the tenant trust relations are required in authorizing cross-tenant accesses, the other way to revoke a cross-tenant AA is removing all the TT it depends on by the respective trustors (builder of the trust relation) who are the permission owners in both Type-α and Type-γ trust or the user owner in Type-β trust. Note that by removing a trust relation, all of the authorization assignments depending on the

[9]One could consider allowing TT to include a mix of the $\trianglelefteq$ relations but this is likely to be confusing and overkill.

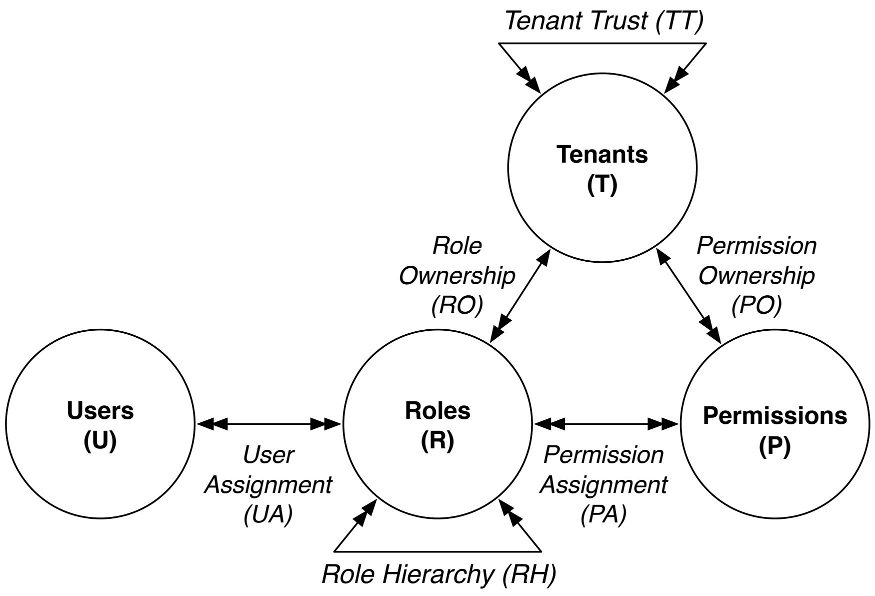


Figure 3.6: Role-Based Cross-Tenant Trust Model

particular trust relation are automatically revoked[10]. Moreover, removing a trust relation does not affect intra-tenant assignments.

3.3.5 Role-Based CTTM

Role-based access control (RBAC) models [27, 48] have been utilized by enterprise information systems for decades. The introduction of roles intermediates the authorization assignments from users to permissions and easies administration of access control policies. The benefit of roles is also applicable to CTTM. Due to the on-demand self-service feature of the cloud, managing the authorization assignments from users directly to permissions is subject to dynamic changes of users and tenants. Therefore, we propose a reasonable extension of CTTM, role-based CTTM (RB-CTTM) in which each user can have different roles in different tenants and a role belongs to a single tenant so that a change of the user does not affect the entire authorization assignment.

The RB-CTTM model, as shown in Figure 3.6, contains four entity components: *users* (U), *roles* (R), *permissions* (P), and *tenants* (T). The definition of T and P are identical to those in

[10]For simplicity and security at the model level, we assume that the corresponding cross-tenant assignments are automatically revoked as soon as the trust relation is removed. Depending upon implementation, the assignment issuer may also choose to manually clear or even keep the nonfunctional hanging cross-tenant assignments for future use.

CTTM while the definition of U is changed. Users are no longer owned by tenants but the roles are, while users are members of roles. RO depicts the many-to-one ownership relation from R to T.

USERS. A user is a global identity of an individual or a process. It is authenticated as a federated ID [19] which is globally unique for all the tenants in the cloud service. A user can be assigned to multiple roles in multiple tenants. In the car rental example, Bob is a user with the student role in $UTSA$. At the same time, he could also be a member of the customer role in $AVIS$. In this way, having different roles in different tenants does not change the user identity.

ROLES. A role is a job function associated with a tenant. A role belongs to a single tenant while a tenant may own multiple roles. Basically, a role is a pair of role name and tenant and is written in the format of "$role\_name\#tenant$". Sticking with the car rental example, the student role in $UTSA$ is noted as $student\#UTSA$ which is not associated with any tenant other than $UTSA$.

The formal definition of RB-CTTM follows.

Definition 12. *The role-based cross-tenant trust model (RB-CTTM) has the following components.*

- *T, P, TT and PO are unchanged from* CTTM*; U and R are finite sets of global users and roles respectively;*

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

- *$UA \subseteq U \times R$, is a many-to-many user-to-role assignment relation;*

- *$PA \subseteq P \times R$, is a many-to-many permission-to-role assignment relation requiring that $(p, r) \in PA$ only if*
  $permOwner(p) \equiv roleOwner(r) \vee$
  $permOwner(p) \trianglelefteq_{\alpha} roleOwner(r) \vee$

$roleOwner(r) \trianglelefteq_\beta permOwner(p) \vee$

$permOwner(p) \trianglelefteq_\gamma roleOwner(r)$,

where only one of the $\trianglelefteq$ requirements can apply depending on the nature of TT;

- *$RH \subseteq R \times R$ is a partial order on R called role hierarchy or role dominance relation, also written as "$\geq$", requiring that $r_2 \geq r_1$ only if*

  $roleOwner(r_1) \equiv roleOwner(r_2) \vee$

  $roleOwner(r_1) \trianglelefteq_\alpha roleOwner(r_2) \vee$

  $roleOwner(r_2) \trianglelefteq_\beta roleOwner(r_1) \vee$

  $roleOwner(r_1) \trianglelefteq_\gamma roleOwner(r_2)$,

  where only one of the $\trianglelefteq$ requirements can apply depending on the nature of TT.

Note that in order to enable role activation, a session entity component and corresponding functions could also be added to RB-CTTM like those in RBAC. However, we do not discuss session here since it is not the core idea of our proposal.

Since the users are global in RB-CTTM, UA is an arbitrary relation without limitation of tenants, unlike RH and PA. Both RH and PA are tenant-aware assignments with can be intra-tenant or cross-tenant. Intra-tenant RH and PA are similar to those with the same names respectively in RBAC models [27]. Each cross-tenant assignment requires at least one appropriate trust relation as specified in Definition 12.

With RH senior roles inherit permissions from their junior roles so that the permissions are transitively inherited to the users. We can authorize a cross-tenant access from a role to a permission in different tenants through either RH or PA. But, it is worth noting that in a Type-γ trust relation, cross-tenant PA is not recommended to be allowed. If the trustor (the permission owner) allows the trustee (the role owner) to make cross-tenant PA, then the trustor will never know which trustee's roles inherit the permissions, let alone controlling the inheritance. A better practice could be making only intra-tenant PA and cross-tenant RH. In this way, the trustor can at least control the inheritance of its own permissions by PA.

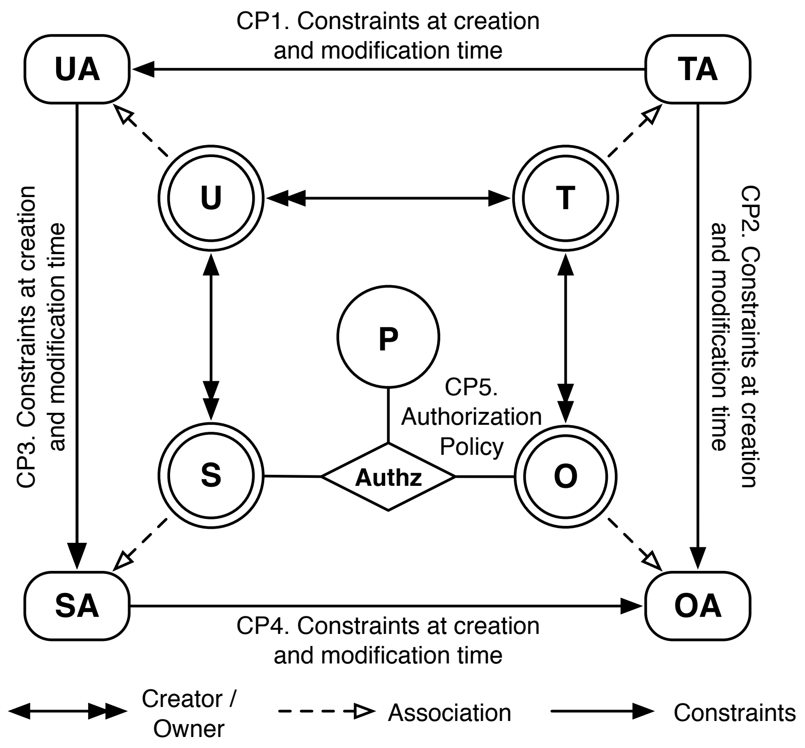

CP1. Constraints at creation and modification time
UA
TA
U
T
CP3. Constraints at creation and modification time
CP2. Constraints at creation and modification time
P
CP5. Authorization Policy
S
Authz
O
SA
OA
CP4. Constraints at creation and modification time
Creator / Owner
Association
Constraints

Figure 3.7: Multi-Tenant Attribute-Based Access Control (MT-ABAC) Model

3.4 MT-ABAC

In this section, we formally introduce the MT-ABAC model, explain the authorization process for multi-tenant accesses and configure it to cover MT-RBAC [52] model.

3.4.1 MT-ABAC Components

The core components of MT-ABAC model, as shown in Figure 3.7, are: Tenants (T), Users (U), Subjects (S), Objects (O), Tenant Attributes (TA), User Attributes (UA), Subject Attributes (SA), Object Attributes (OA), Permissions (P), authorization policies and constraint policies for creation and modifications of TA, UA, SA, and OA.

Tenants represent isolated operation domains leased by a cloud service consumer. In real-world clouds, tenants may also be regarded as "domains" or "accounts" etc. A tenant is also an acting entity who creates users and objects of its own and has ultimate authority within its scope.

A tenant may represent a company, a project team, or even an individual.

Users are identifiers for acting entities inside a tenant. A user can be associated to an employee of the company, a team member of the project, or even the individual himself who owns the tenant. A user is created and managed by a single tenant who may own multiple users. The existence of the users relies on the owning tenant.[11] If a tenant is removed, the users inside the tenant will be deleted automatically.

Subjects represent access sessions of corresponding users to objects. A user can access objects only through the subjects created by himself. In practice, a subject embodies itself as a credential issued by the identity and access management services [7] in the cloud while its attributes, such as assigned roles, tenant, or clearance of the particular access, are also contained in the credential.

Objects are pieces or collections of the cloud resources inside a tenant. Typical objects are virtual machines (VMs) or VM imagesin IaaS, application instances in PaaS and units of user data in SaaS. In multi-tenant schemes, an object belongs and only belongs to a single tenant.

Permissions represent privileges for subjects accessing objects. Since the object types are typically consistent in a single cloud, the object access methods can be easily unified, such as create, read, update and delete (CRUD) operations.

Authorization policies are functions evaluating each access decision. An authorization policy takes subject, object and requested permissions as parameters and returns an authorization decision based on correlated attributes. The decision yields to one of the following: “allow”, “deny” or “don’t know” which indicates the decision cannot be reached for many reasons including condition not applicable, necessary attributes not found or conflict of multiple policies, etc.

Constraints are conditions for the creation and modification of UA, SA and OA. The change of attributes will happen only if the conditions are satisfied. The constraints also apply to the outcome of the changes.

[11]In real world cloud systems, such as OpenStack [6], the situation may vary. A user may be independent to a tenant. OpenStack requires the capability to create and manage subjects associated to a tenant in order to get access to the objects inside the tenant.

3.4.2 Formal MT-ABAC Model

The formal definition of MT-ABAC follows.

Definition 13 (Base Model)**.** *The base model of MT-ABAC has the following components:*

- T*,* U*,* S*,* O *and* P *are finite sets of tenants, users, subjects, objects and permissions respectively;*
- TA*,* UA*,* SA*, and* OA *are finite sets of functions representing tenant attributes, user attributes, subject attributes, and object attributes respectively;*
- $Scope : TA \cup UA \cup SA \cup OA \to \{atomic\ values\}$*, an attribute scope function returning a finite set of atomic values;*
- $attrType : TA \cup UA \cup SA \cup OA \to \{set, atomic\}$*, an attribute type function mapping each attribute to its type which is either set or atomic;*
- $utid : U \to T$*, an atomic attribute in* UA *mapping each user to its owning tenant;*
- $otid : O \to T$*, an atomic attribute in* OA *mapping each object to its owning tenant;*
- $sowner : S \to U$*, an atomic attribute in* SA *mapping each subject to its owning user;*
- $trustees : T \to 2^T$*, a set attribute in* TA *mapping each tenant to its trustee tenants;*
- *CP[1-5] are policy configuration points for constraints and authorization policy functions;*
- $cstrTU : T \times U \to \{True, False\}$*, a constraint policy function at CP1 requiring* $t \in T \wedge u \in U \wedge u.utid \equiv t$*;* [12]
- $cstrTO : T \times O \to \{True, False\}$*, a constraint policy function at CP2 requiring* $t \in T \wedge o \in O \wedge o.otid \equiv t$*;*

[12]The "." operator is used to denote the attribute function given the first operand as an entity and the second operand as an attribute name. The function returns the corresponding attribute value.

- *$cstrUS : U \times S \rightarrow \{True, False\}$, a constraint policy function at CP3 requiring $u \in U \wedge s \in S \wedge s.sowner \equiv u$;*

- *$cstrSO : S \times O \rightarrow \{True, False\}$, a constraint policy function at CP4 requiring $s \in S \wedge o \in O \wedge s.sowner.utid \in o.otid.trustees$; and*

- *$canAccess : S \times O \times P \rightarrow \{True, False\}$, an authorization policy function at CP5 requiring $s \in S \wedge o \in O \wedge s.sowner.utid \in o.otid.trustees$.*

All the four entity components, T, U, S and O as shown double circled in Figure 3.7, have associated attributes. T is the set of existing tenants [13] and TA is the set of attributes for tenants in T. Each attribute in TA maps a tenant in T to a corresponding attribute value which is either set or atomic depending on the $attrType$ of the specific attribute. Similarly, U, S and O are associated with UA, SA and OA respectively as specified in Definition 13.

Some special attributes are mandatory in the MT-ABAC model. As a special attribute in UA, $utid$ maps each user to its owning tenant. A user, in its lifetime, is associated with its owning tenant so as an object. The owner of an object is specified with the $otid$ attribute in OA. The situation for subjects is a little different. A subject is created by its owning user, specified with $sowner$. Note that a subject may be able to access an object from a tenant different with its owning user's owning tenant while the appropriate trust relation between the two tenants exists as specified in $cstrSO$ and $canAccess$. Constraint $cstrTU$ and $cstrTO$ require the accessing tenant's ownership of users and objects respectively. In the case of multi-tenancy, each user and object belongs to a single tenant. Each user can create multiple subjects of its own while each subject is only controlled by its owning user as specified in the prerequisite of the constraint $cstrUS$.

The trust relation is indispensable in MT-ABAC in order to control cross-tenant accesses. For simplicity, we only use Type-γ trust [53] in the MT-ABAC model which can be easily extended to support multiple types of trusts by adding trust-type-specific "trustees" attribute, such as "α-trustees", "β-trustees" and "γ-trustees". Figure 3.8a shows an example of intra-tenant access re-

[13]The creation and deletion of tenants are controlled by the cloud service provider (CSP) either manually or automatically. Users and objects are controlled by the owning tenants. Subjects are controlled by the owning users.

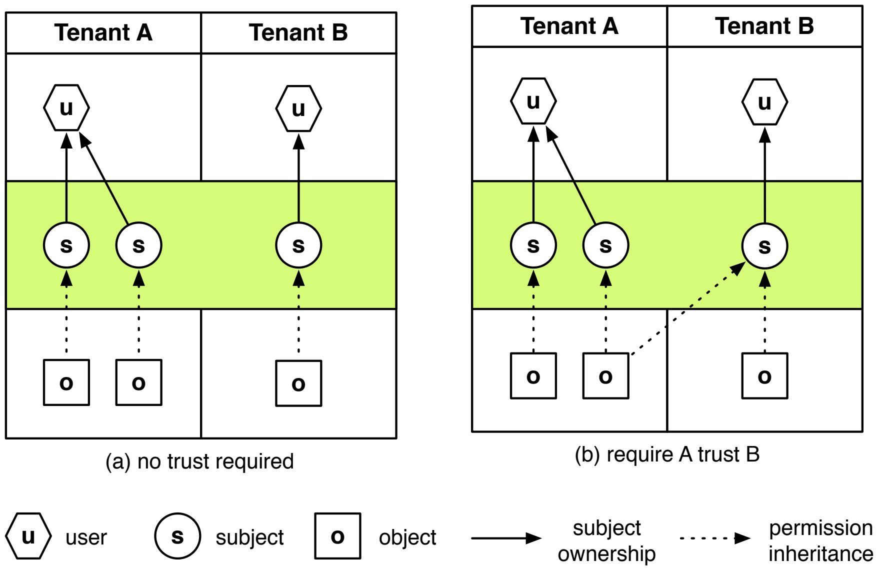


Figure 3.8: Multi-tenant accesses with Type-γ trusts

quiring no trust (tenants trust themselves by default). Figure 3.8b illustrates how a user from a trustee tenant can obtain cross-tenant access to the object in the trustor tenant. The trust relation is always established by the trustor and referred during the process of authorization decision.

Note that a subject is not tenant-specific since its $sowner$ attribute indirectly indicates its relationship with a single tenant. In fact, a subject is only associated with its owning user and the accessing object both of which are tenant-specific. Only if the subject's tenant is trusted by the object's tenant, the access request will be evaluated with other authorization policies; otherwise, the request will be denied. Formally,

$$s.sowner.utid \in o.otid.trustees \tag{3.9}$$

can be extended to address all the three types of trust relations as the following.

$$s.sowner.utid \in o.otid.\alpha{-}trustees \tag{3.10}$$

$$o.otid \in s.sowner.utid.\beta{-}trustees \tag{3.11}$$

$$s.sowner.utid \in o.otid.\gamma{-}trustees. \tag{3.12}$$

Policy Configuration Points

In MT-ABAC, there are totally five policy configuration points (PCP). CP1 through CP4 specify the constraints for the creation and modification of attributes controlling the state transitions of each entity. CP5 specifies the authorization policies based on attributes depending on the accessing state. Even if the policies at different configuration points may be specified by different authorities, their policies may affect the authorization decision. The constraint policies are specified by the CSP since some cloud-level constraints define the access control model for all the tenants, such as cardinality and separation of duty. Meanwhile, some constraints are customizable by tenants in CP3 and CP4, because users and objects are owned by tenants who should be able to control the access behavior between them. The authorization policy is maintained by each tenant as well as some tenant attributes, such as the *trustees* attribute in TA.

Typically, the policy configuration languages (PCL) in all the PCPs are common in format and logical structure [36] so that they can be easily compiled together during policy decision time. In some cases, external PCPs may require PCL translation or mapping for making proper decisions. On top of the potential PCL translation problem, the policy resolution algorithm is also critical in the policy decision process. Similar problems have been identified and solved in multi-domain literature [49,50]. Since the multi-tenant cloud environment does not distinguish the problems from the previous ones, we treat them as solved problems with no more discussion in this dissertation.

Meta-Attribute

Meta-attributes [35] are not included in the base model but useful specifying meta information associated with attributes. In another word, meta-attributes are attributes of attributes. Thus, a meta-attribute is also an attribute function given the scope of the associated attribute as a parameter. The type of a meta-attribute is either set or atomic. Moreover, a meta-attribute may also have meta-attributes forming cascading structure of meta-attributes. For example, in multi-tenant environment if a role as a user attribute is only valid in a tenant, we call it a tenant-specific role which can be captured by the tenant meta-attribute associated with the role attribute. In this way, the tenant meta-attribute links the role to the attributes of the tenant. For convenience, we still use the "." operator to note the relation between an attribute and its meta-attribute. Thus, a user u's role attribute $u.urole$ may be associated with cascaded meta-attributes, such as $u.urole.roleOwner.trustees$ conveying the trustees of the role's owing tenant.

Attribute Mapping

Even though most of the attributes tend to be common for all the tenants, the needs to allow customized attributes in each tenant may lead to the attribute mapping problem between tenants. The attributes, such as name, gender, date of birth and creation time, are absolute values and usually uniform across all the tenants. Others, like roles, department and classification, are more likely to be tenant-specific. For cross-tenant accesses, the policy decision may become unreliable if the remote and local attributes are not properly mapped to a uniform format or value.

There are two major factors of attributes need to be mapped in cross-tenant scenarios, the attribute name and the attribute value.The direction of mapping is always from the remote tenant to the local tenant in terms of the object to be accessed. In other words, the attributes are mapped from the user's owning tenant to the object's owning tenant. The mapping policy can be specified by either tenant before cross-tenant accesses happen. It is straightforward to map the attribute names between tenants since each tenant may give different names to a same attribute. For example, in some tenants the roles attribute is name "roles" in UA while in other tenants it is named "groups" instead following their convention. In cases like this, the remote attribute name is directly associated with the local corresponding name.

In cases of attribute value mapping, the situation is much more sophisticated since the types and ranges of the attribute values may not be the same in the local and the remote tenants. For instance, an "age group" attribute may be either set or atomic and the ranges may vary in different tenants. Furthermore, a same value of an attribute may be perceived differently in the two tenants.

For example, the “roles” attribute may have a “manager” value which represents different levels of job positions in different organizations. As a result, in order to express the complex mapping relations between attribute values, we need a systematic mapping solution which is recognized here but beyond the scope of this dissertation.

3.4.3 MT-RBAC Configuration

As a unified model, MT-ABAC can be configured to various MTAC models including MT-RBAC. In this section, we present the MT-RBAC configuration using MT-ABAC.

The roles in MT-RBAC are tenant-specific and associated with users, sessions and permissions. In MT-ABAC, roles are presented as set attributes in UA, SA and OA. When assigned, each role has a meta-attribute of its owning tenant, namely “$roleOwner$”. The user-role assignments are represented by the role attributes in UA. For simplicity, we assume there are two permissions, read and write available for role-permission assignments which are represented by the “rroles” and the “wroles” respectively in OA. The roles assigned to a user may be associated with different tenants who trust the user’s owning tenant (to make the assignments). The role hierarchy assignment is specified as a set attribute “role_hrchy” in TA and maintained by each tenant. Hence, the constraint policies in CP1 should contain the following requirement.

$$u.utid \in u.urole.roleOwner.trustees \tag{3.13}$$

and

$$\forall r' \leq r : r.roleOwner \in r'.roleOwner.trustees. \tag{3.14}$$

The roles of a subject is always a subset of its owning user’s roles. This constraint is specified in CP3 as the following.

$$s.sroles \subseteq s.sowner.uroles. \tag{3.15}$$

Since cross-tenant permission-role assignments are not allowed in MT-RBAC, the “rroles” and the“wroles” of an object should contain only roles belong to the object’s owning tenant. Moreover, all the roles of a subject and their junior roles should belong to tenants who are trusted by the object's owning tenant. The following constraints should be configured in CP4.

$$\forall rr \in o.rroles, wr \in o.wroles : rr.roleOwner \equiv wr.roleOwner \equiv o.otid \tag{3.16}$$

and

$$\forall r \in s.sroles, r' \leq r : r'.roleOwner \in o.otid.trustees. \tag{3.17}$$

3.5 Discussions

The formal role-based and attribute-based MTAC models are presented above. In this section, the comparison of role-based trust models, the constraints needed in multi-tenant collaborations and the trusts supported in real-world clouds are discussed.

3.5.1 Role-Based Trust Models

Role-Based Trust Models are effective in terms of access control for collaboration in distributed environments. Trust relations, in certain forms, are established among administrative domains for sharing. The effects of different trust models vary in different aspects. We hereby identify some crucial differences among three role base trust models: Role-based Trust-management (RT) framework [40], Multi-Tenancy Authorization System (MTAS) [22, 55], and our new approach MT-RBAC in this dissertation, as shown in Table 3.3.

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

Table 3.3: Trust Model Comparison. A and B represent two entities in RT or tenants in MTAS and MT-RBAC. A represents the resource owner and B the requester.

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

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

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

Last but not least, we discuss the necessity of a prerequisite to enable collaborations with role based trust models. Common vocabulary is a term introduced in RT [40] requiring both collaborators of a trust relation to use a mutually understandable definitions of roles. Thus the semantic mismatch issue in decentralized collaboration is mitigated. However, there is no such requirement in MTAS and MT-RBAC since their trust relations allow exposure of the truster's roles to the trustee so that the definition of roles is perceivable to the administrator within a common centralized facility such as an AaaS platform.

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

3.5.2 Constraints

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

Cyclic Role Hierarchy. A "role cycle" may be formed across tenants in MTAS systems without proper constraints. This may lead to violation of the security principal [31] in interoperation. Similar problems in secure interoperation have been addressed in multi-domain environment [49]. Some computational challenges discussed in [31] remains 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. During collaborations with MTAS, we identify two levels of separation of duties (SoD), issuer level and role level. For issuer level SoD, one collaborating issuer cannot execute two conflict responsibilities. For instance, SOX [10] 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 1.1, 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 tenant. Also, the conflict duties may be associated with different cloud services. In this case, the constraint policy composition process can become very complicated [16]. 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 among issuers also needs to be managed. For example, two competing issuers should not be trusted by a single issuer so that the security and privacy of the trustee issuer's sensitive information are protected against the competitors. This situation is already abstracted and addressed by the Chinese Wall model [21] which can be integrated in the centralized AaaS platform to avoid conflict of interests. Essentially, the issuers are grouped into "conflict of interest (COI) classes" and by mandatory ruling all issuers are allowed to trust at most one issuer belonging to each such conflict of interest 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 conflict of interest issuers.

3.5.3 Trusts in 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 cloud service providers, allows its user to establish trust between two accounts, for example Production and Development, in Amazon Web Services (AWS) in order to simplify user management between Production and Development and many other use cases [9]. Account is comparable with issuer in MTAS or tenant in MT-RBAC. 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 account 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 specify an authorization assignment allowing only developers, as one of the roles, from the Development account to access some of its resources. As described above, the AWS account trust does not provide dual control feature and some offline communications, such as the role names, are required.

OpenStack is an open source cloud computing platform for public and private clouds [5]. The trust mechanism inside OpenStack is built on the basis of user-level delegation. Use the same Production and Development account example. The concept of account or issuer are 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 [52,53,55]. 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.

Chapter 4: MULTI-TENANT AUTHORIZATION AS A SERVICE (MTAAAS)

Now that both of the role-based and attribute-based MTAC models have been proposed in Chapter 3, we move our focus to the lower enforcement layer. Particularly, enforcement refers to the architecture of the authorization service which is capable of accommondating all the MTAC models in the cloud environment. In this chapter, we introduce the MTAaaS architecture, the policy specification of MTAC models in XACML and a prototype system evaluating the performance and scalability of the proposed models.

4.1 MTAaaS Architecture

In order to foster fine-grained access control in collaborative cloud environments, we propose Multi-Tenant Authorization as a Service (MTAaaS) as a novel service model providing an independent authorization infrastructure in a multi-tenancy manner. This service can be integrated with the existing cloud services to manage and process authorizations for them. MTAaaS is compatible with all the MTAC models as described in Chapter 3.

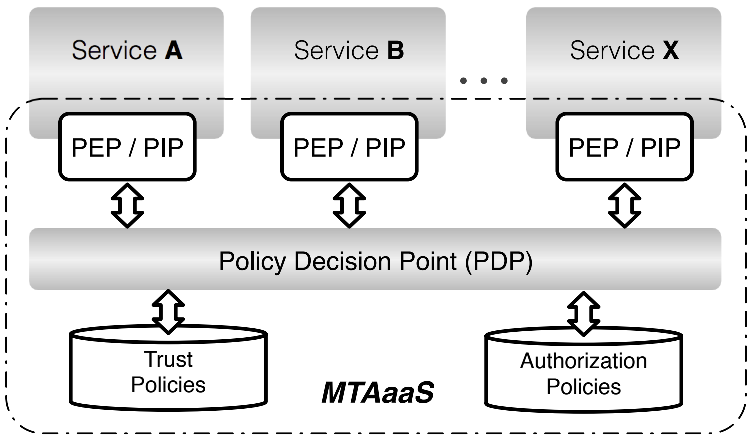


Figure 4.1: MTAaaS Architecture

Figure 4.1 illustrates the architecture of the MTAaaS in which the cloud services use a common authorization service (i.e., MTAaaS) by associating each service with a distributed PEP/PIP module in a multi-tenant way. The PEP/PIP module is a combination of a policy enforcement point (PEP) and a policy information point (PIP). The PEPs parse the requests from end users and generate normalized (XACML format) requests to the centralized policy decision point (PDP) module which refers to authorization and trust policies stored in the centralized policy repository.[1] The PIPs collect attributes and other context information related with authorization and make them available for the PDP. After the decision is made, an XACML format response will be sent back to the requesting PEP to take effect with respect to the requested access. The integrity and authenticity of communication messages should be guaranteed, say via long-lived TLS connections between PEPs and the central PDP. For simplicity, they are not included in the prototype implementation. The prototype can be extended to a cloud authorization service with distributed PDPs.

4.2 Policy Specifications

In order to demonstrate the feasibility of the MTAC models, we give the policy specification here in extensible access control markup language (XACML) [12]. The normative specification of RBAC policies with XACML2.0 language has been proposed by OASIS XACML TC [11]. Its Role PolicySet (RPS) and Permission PolicySet (PPS), representing UA and PA respectively, are also used with the MTAS and MT-RBAC policy specifications. Additionally, a novel Trust PolicySet (TPS) is added to express the trust relation, as illustrated in the following.

In order to accommodate various attributes, the MT-ABAC policy specification uses the default structure of XACML since it is designed to express attribute-based policies. Similar to role-based models, the TPS is added in each tenant's policy sets to specify the trust relations.

4.2.1 MTAS Policy Specification

In order to better explain how MTAS XACML policy work in the MTAaaS architecture, we develop an example policy structure as shown in Figure 4.2 using the out-sourcing example as described in

[1]Note that the locations of the policies may vary in different implementation. Policy discovery mechanisms may be needed in distributed environments and is left as an implementation issue.

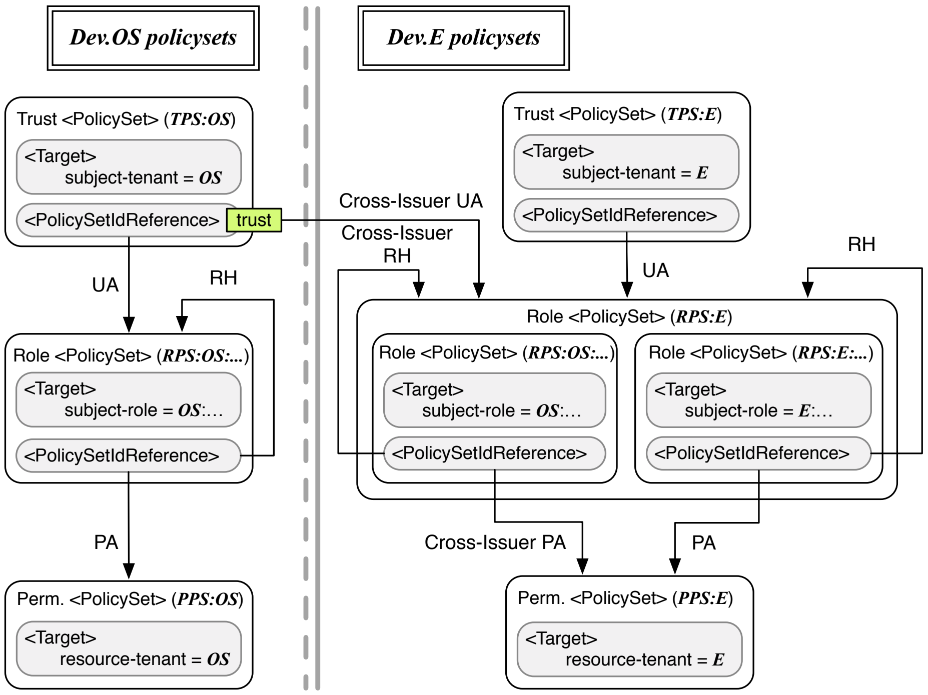

Dev.OS policysets
Dev.E policysets
Trust <PolicySet> (TPS:OS)
<Target> subject-tenant = OS
<PolicySetIdReference>
trust
Cross-Issuer UA
Cross-Issuer RH
Trust <PolicySet> (TPS:E)
<Target> subject-tenant = E
<PolicySetIdReference>
UA
RH
UA
RH
Role <PolicySet> (RPS:OS:...)
<Target> subject-role = OS:...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

Figure 4.2: Example MTAS policy structure with trust relation $OS \lesssim E$ as highlighted.

Figure 1.1.[2] 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 since the subject tenant attribute in the request has value OS. According to 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 $RPS{:}OS$ since 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 role hierarchy 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

[2]For simplicity, we use E and OS to represent tenants $Dev.E$ and $Dev.OS$ respectively in the example policies.

responded to PEP who will grant 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 trust relation above, 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-tenant PA and RH respectively will return a permit for the request.

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

4.2.2 MT-RBAC Policy Specification

We specify MT-RBAC policies in extensible access control markup language (XACML). According to the normative specification of RBAC policies [11], we keep using the Role PolicySet (RPS) and the Permission PolicySet (PPS), representing UA and PA respectively in MT-RBAC. Additionally, a novel Trust PolicySet (TPS) is added. In order to express MT-RBAC$_1$ and MT-RBAC$_2$ policies, we also introduce $TIPR$ PolicySet (IPS) and $TDPR$ PolicySet (DPS). In this section, we present an MT-RBAC$_2$ policy example and the corresponding authorization process.

Cross-Tenant UA: starting from the upper RPS in Figure 4.3, a user in te sends a request to access tr's resource. If the user is already assigned to a role in tr, RPS will forward the request to tr's TPS who checks if $tr \trianglelefteq te$. If the trust relation does not exist, the request will be denied; otherwise, tr's DPS will check if the user's role in tr is in $\mathcal{P}_{TD}(tr, te)$. If the role is public to te, tr's PPS associated with the role will check the resources and actions in the request. If these do not match the policy rules, the request will be forwarded to the DPS again to verify the visibility of the junior roles to te. This process will execute recursively until a match in PPS rules is found or the lowest role visible to te in tr's role hierarchy is reached. If a match is found, the PDP will respond with a permit, otherwise the PDP will check other authorization paths in the policy tree for a match. If finally no match is found, a deny response will be returned to the PEP.

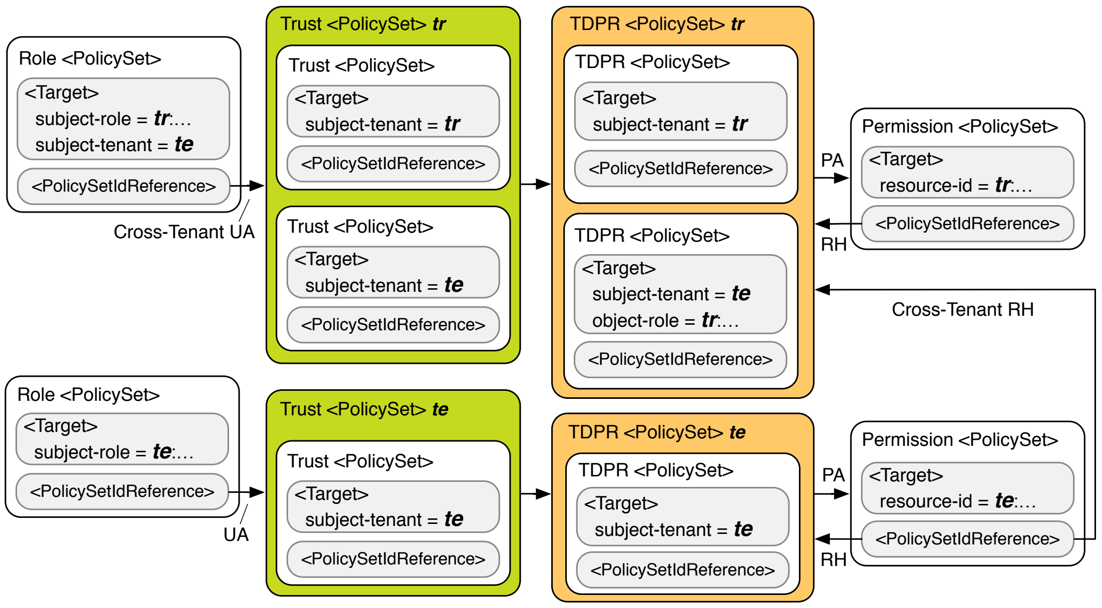


Figure 4.3: The MT-RBAC$_2$ policy enables two independent authorization paths: cross-tenant user assignment (*UA*) and cross-tenant role hierarchy (*RH*), where *tr* trusts *te*.

Cross-Tenant RH: starting from the lower RPS in Figure 4.3, the request from a user of a te's role will first be sent to te's TPS. The te's TPS and the subsequent DPS will forward the user's request because it is an intra-tenant request. Then, the request will arrive to the PPS of the user's role in te. The recursive process of retrieving a proper junior role will take place. Since a cross-tenant RH assignment to a junior role in tr exists, the request will be forwarded to tr's DPS during the recursive process. The steps afterwards is similar as described in the other authorization path.

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

4.3 Prototype and Evaluation

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. Since 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 due to 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 [3]. The PDP is installed on a 64-bit Linux CentOS 6 system with 2.5 GHz dedicated CPUs. The PEPs are built upon SmartMachines [3] 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 4.4. We used eight PEP servers with the same flavor to send concurrent requests to a centralized PDP server which is equipped with various hardware 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. It is deployed on a SmartMachine with SmartOS 1.8.4, 1 GB RAM and shared CPUs. The testbed architecturally simulates an authorization service implementing the MTAS model and supports controlled experiments in a commercial-standard cloud system.

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. Since 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

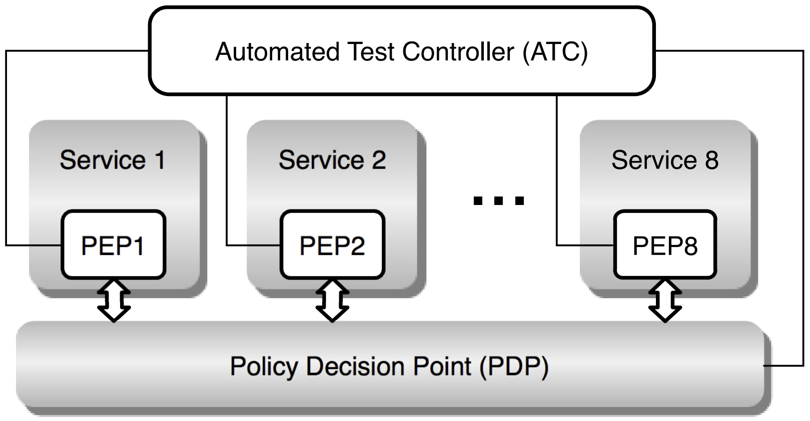
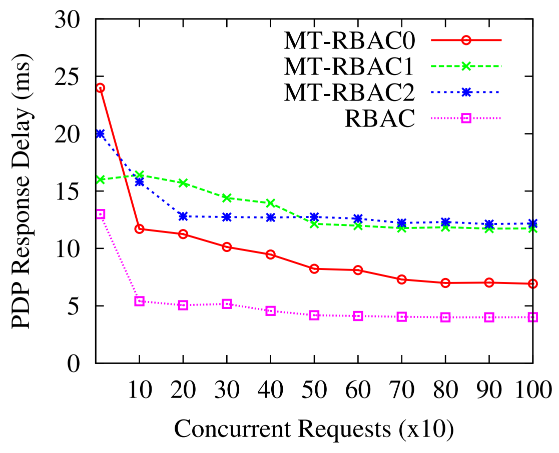


Figure 4.4: MTAaaS Testbed Architecture

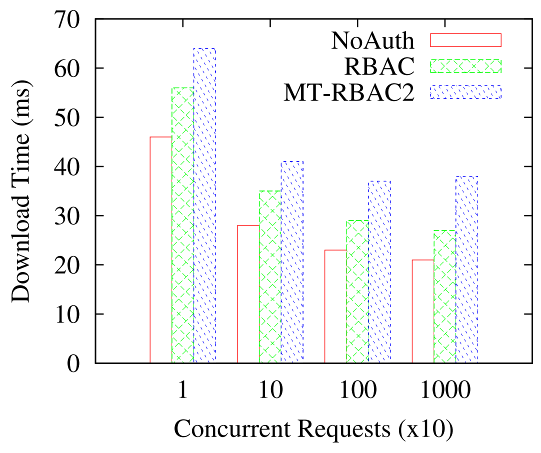
than value of CPU frequency, is regarded as proportional to hardware capability. For example, a PDP with 2GB RAM and 2 Core CPU is considered as of 2 unit computing capacity which doubles hardware capability of its 1 unit counterparts. In our experiments we have PDPs running on 1 unit, 2 unit and 4 unit servers respectively. At the PEP end, we have 8 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 metics 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 takes policy decision time at the PDP end. We call this effect authorization overhead which is inevitable.

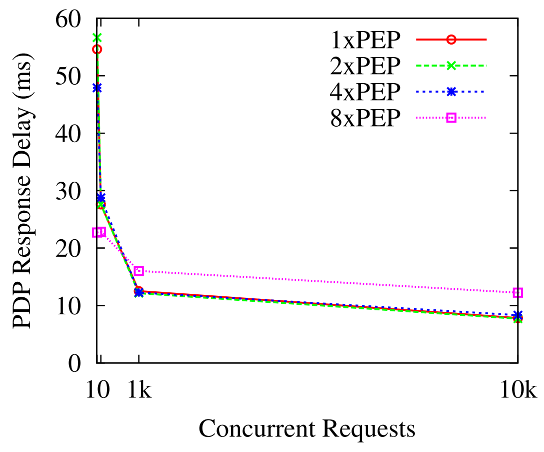
Figure 4.5a compares the policy evaluation delay in RBAC and the three MT-RBAC models. Note that in RBAC cross-tenant access requests are not supported. Due to the caching mechanisms of the operating system, as the number of concurrent requests increases, the average policy decision delay decreases dramatically until it reaches a stable state. RBAC has the least delay of 4.01 ms, while MT-RBAC$_0$ has 6.92 ms delay. The evaluation of TP policies contributes to the extra delay of MT-RBAC$_0$, compared to RBAC. Since IPS and DPS evaluations incur the similar I/O operations to TP evaluations, the authorization delay for MT-RBAC$_1$ and MT-RBAC$_2$ are similar.



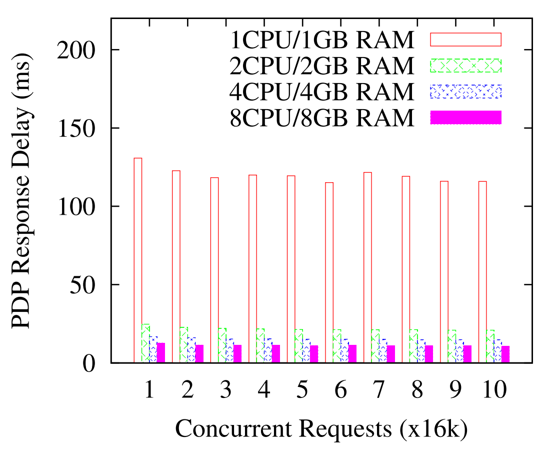
(a) PDP Performance



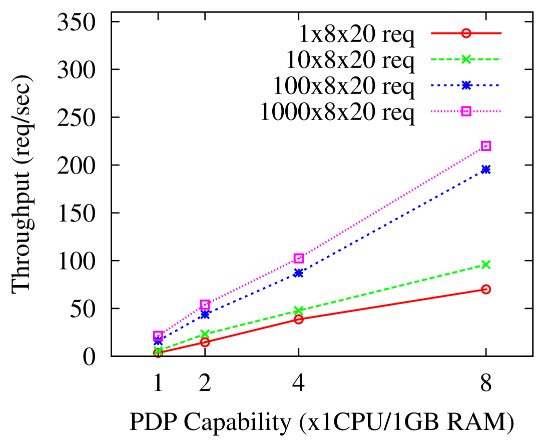
(b) Client-End Performance



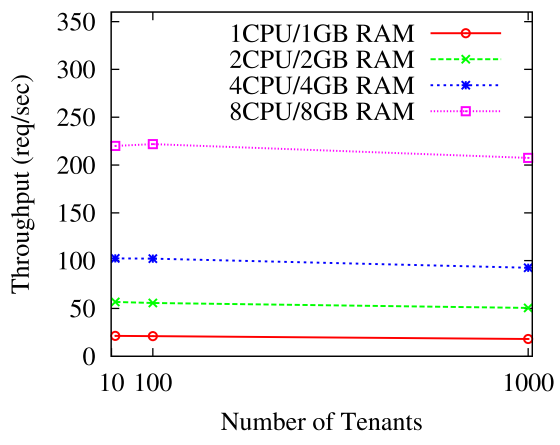
(c) PDP Response Delay with various PEP amount



(d) PDP Response Delay with various hardware capability



(e) Scalability Results with 10 tenants



(f) Policy Complexity Scalability Results

Figure 4.5: Performance and scalability evaluation results

MT-RBAC$_1$ and MT-RBAC$_2$ have the most delay of 11.75 ms and 12.18 ms, respectively. MT-RBAC models introduce acceptable evaluation overheads compared to RBAC.

Figure 4.5b shows a comparison of delays at the client-end of the CloudStorage service. The delays are observed upon a 1KB file downloading task with or without authorization through RBAC or MT-RBAC$_2$. According to the experiment result, we observe that MT-RBAC$_2$ introduces around 12 ms authorization delay which we believe is acceptable for file downloading tasks in cloud storage services.

We evaluate the authorization overhead of our prototype system with MTAS by sending concurrent sample requests from various numbers of PEPs to a 4-unit PDP. We use ten disjoint pairs of sample requests and responses representing intra-tenant and cross-tenant 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. Since 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 4.5c, 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 4.5d illustrates the performance of PDP on different servers with 1, 2, 4 and 8 units of hardware capacity respectively. The experiments are conducted with 8 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 needs 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 the number of tenants as 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.

$$Throughput = \frac{1}{Average\_Delay \times CPU\_Utilization} \tag{4.1}$$

The results shown in Figure 4.5e 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 4.5f 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 inverse proportional with the throughput. Consequently, it is reasonable to believe that the prototype is scalable in the cloud environment.

4.4 Discussions

The centralized architecture of MTAaaS may suffer from efficiency issues because each authorization request needs to be handled by the centralized PDP before the user can access the requested cloud resources. The implementation solution in OpenStack is distributing the PDP into each cloud service so that the authorization decision can be made inside each service. However, this kind of solutions also has drawbacks. Firstly, the complexity of policy management increases tremendously since the policies should be located close to the distributed PDPs for performance reason. Then, the consistency and security of policy become problems. Secondly, the distributed PDPs need to be synchronized in order to keep the consistency of authorization decisions. The synchronization process turns out to be a management overhead. Last but not the least, the identity service issue credentials for authenticated users so that the cloud services can verify their identity. To maintain the PKI-based credentials, all the cloud services need to talk to the identity service intermittently to update their copies of the credential revocation list which is checked for every verification. The cost of maintaining the PKI mechanism is also inevitable. As a result, the distributed PDP solution also suffers from efficiency problems.

Chapter 5: OPENSTACK DOMAIN TRUST IMPLEMENTATION

In previous chapters, we have discussed the policy layer and the enforcement layer models for MTAC. In this chapter, we present our work at the implementation layer, particularly in OpenStack [6], an open-source cloud platform. The general concept of a tenant in a cloud maps to the concept of domain in the Havana release of OpenStack.[1] We propose a domain trust implementation enforcing multi-domain access control, which is essentially the MTAC models integrated with OpenStack.

5.1 Background and Motivation

The identity service in OpenStack, called Keystone, is used to manage users as globally available resources. More specifically, the administrator of a domain can view all the user information and assign any user to roles controlled by that domain. Each user, as created, belongs to a single domain and the domain owner or administrator can only see and manage users within the domain. So far, the use cases of cross-domain access have not been carefully addressed in OpenStack.

In this section, we use a DevOps [1] example to explain why we need cross-domain accesses in the cloud and the potential security problems without proper control. Also, we discuss the pros and cons of existing cross-domain authorization solutions.

Motivation

DevOps is a newly emerged software development methodology that stresses collaboration among software development, quality assurance (QA) and operations. Numerous companies are actively practicing DevOps since it aims to help organizations rapidly produce software products and services [1]. When DevOps for an organization comes into play in an OpenStack cloud, cross-domain accesses become inevitable and requires proper access control. Figure 5.1 shows the authorization

[1]Previous releases of OpenStack employed the term tenant for what has now come to be called project in OpenStack. The term tenant is no longer used in OpenStack. In this dissertation we use the term tenant as a generic concept in cloud computing, while domain is specific to OpenStack as its realization of a tenant.

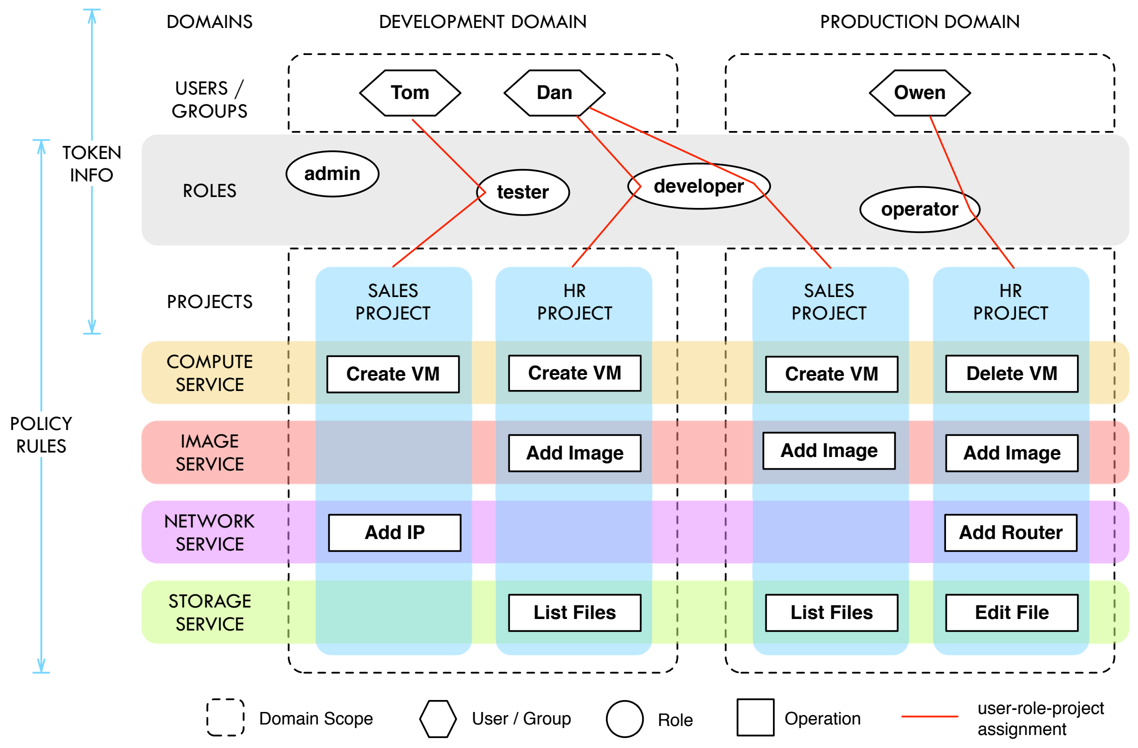


Figure 5.1: An DevOps use case of cross-domain accesses.

related components in OpenStack giving cross-domain accesses for a DevOps use case. The token information is managed by the centralized identity service who issues the tokens for users. The users use their tokens to request access to cloud services. The authorization policies and PDPs are distributed in each cloud service.

Suppose the organization has two domains in the cloud: *Production* and *Development*. *Production* hosts live applications supporting the organization's daily business requiring strict controls on changes. Meanwhile, *Development* consists of development and testing environments, basically a sandbox, where developers and testers can freely conduct experiments with the cloud resources. The isolation of the two domains is mandatory for best practice and compliance reasons. Each domain contains its own set of users, groups, projects and controlled access to the full-spectrum of cloud services, such as compute, image and network. As shown in Figure 5.1, Owen is an operator in *Production*. Dan and Tom are a developer and a tester respectively in *Development*. Each domain has two independent projects: *Sales* and *HR*. The users are assigned necessary permissions to access projects in their owning domains to accomplish their daily jobs. For instance, in a DevOps case the infrastructure of a live application in *Sales.Production*[2] needs to be modified by a developer. Dan as a developer in *Development* assigned that job needs to be authorized to access the application in *Sales.Production*. The *Production* administrator may prefer not to create another user account for Dan in *Production* but to assign Dan a developer role in *Sales.Production* instead, for the following benefits.

- Dan does not have to switch between user accounts in different domains.
- Intra-domain and cross-domain assignments can be distinguished.
- The *Production* administrator can avoid removing the temporary user after the completion of the DevOps case.

This example is a typical use case for organizations using either public or community clouds. By design, OpenStack supports cross-domain assignments however they are treated indifferently than intra-domain assignments. For example, *Production*'s administrator can assign any user from other domains to roles in *Production*'s projects. This approach may cause a series of problems as the following.

a) The *Production* administrator should be able to retrieve the user information and the role assignments in *Development* in order to issue proper cross-domain assignments.

b) The *Development* administrator should be able control the cross-domain authorization.

c) Since DevOps jobs are usually temporary, the management of cross-domain authorization should be flexible.

d) Additional constraints upon cross-domain accesses should be supported.

[2]We use "." to represent the ownership relation between projects and domains. For example, *Sales.Production* refers to the *Sales* project in *Production* domain.

On the one hand, the visibility of a user's roles inside its owning domain provides crucial information for other domain administrators to authorize access of the user since the users in OpenStack are not global but identifiable inside each domain. On the other hand, the user owner needs to monitor or constrain the roles assigned to its users in other domains in order to prevent violations of security principals in multi-domain interoperation [16,49]. In this setting, both of the collaborating domains should have control over the cross-domain access.

The visibility issue in Problem a) can be solved by centralizing the administration of all cross-domain assignments to the cloud administrator but the administrative overhead may become overwhelming. Moreover, it is inappropriate for the cloud administrator to be so involved in the management of individual domains for security and privacy reasons. To address Problem b), mechanisms involving both domain administrators, in other words "dual-control", should be introduced. For Problem c) we need a rapid means to enable or disable cross-domain assignments for better efficiency. Regarding Problem d), constraints like acyclic role hierarchy, separation of duty and conflict of interests need to be addressed in cross-domain access control.

Existing Approaches

We have found similar problems in Microsoft Windows Active Directory (AD). An AD, comparable with the identity service in OpenStack, maintains various types of trust relations allowing users in one domain to access resources in another [4]. But they are not directly applicable in the cloud environment since AD is designed to manage identities for a centralized authority but not decentralized ones like in the cloud.

Currently, OpenStack Keystone supports user-level delegation. In particular, a user can delegate a part of his or her permissions to another user through a trust relation. The trustee can impersonate the trustor to perform a subset of the permissions that the trustor has been authorized. Issuing a trust relation does not require involvement of the domain administrators so that Problem b) still exists. In addition, it requires a single user to have all the permissions that the requesting user needs in the target project and limits the capability of cross-domain collaborations.

Amazon Web Service (AWS) allows delegating access across accounts (accounts are comparable to OpenStack domains). By creating a trust relation and associating an assumed role with it, the trustor account authorizes the users from the trustee account to access permissions associated with the assumed role in the trustor account. In this way, cross-account accesses are enabled. However, the trust relation cannot support customized control other than the assumed role or be constrained.

Scope and Assumptions

In this dissertation we assume that cross-domain authorization only happens in a single cloud. Nevertheless, the model we propose may be extended to federated cloud scenarios. We assume the users in our models are properly authenticated as supported by Keystone. Our discussion and implementation are based on the Havana release of OpenStack [6].

5.2 OpenStack Access Control Model

In this section, we present the core OpenStack Access Control (OSAC) model based on the OpenStack Identity API v3 [7] which is currently the latest stable version. Since OpenStack is a rapidly changing system solving practical problems, we feel it impossible and unnecessary to model every feature in OpenStack identity service. Instead, we keep only the core components in the model and formally present how they interplay with each other in the authorization and administration processes. Hence, the term core OpenStack Access Control model. For simplicity we will often omit the core prefix.

Core OSAC

Core OSAC extends the traditional RBAC model [27] to support multi-tenancy. The model elements and relations are defined in Figure 5.2. OSAC contains eight core entity components: Users (U), Groups (G), Projects (P), Domains (D), Roles (R), Services (S), Operations (O) and Tokens (T). Other entities in the OpenStack Identity API are regarded as implementation specific such as credentials, regions and endpoints. Each of the entities has a globally unique resource identifier

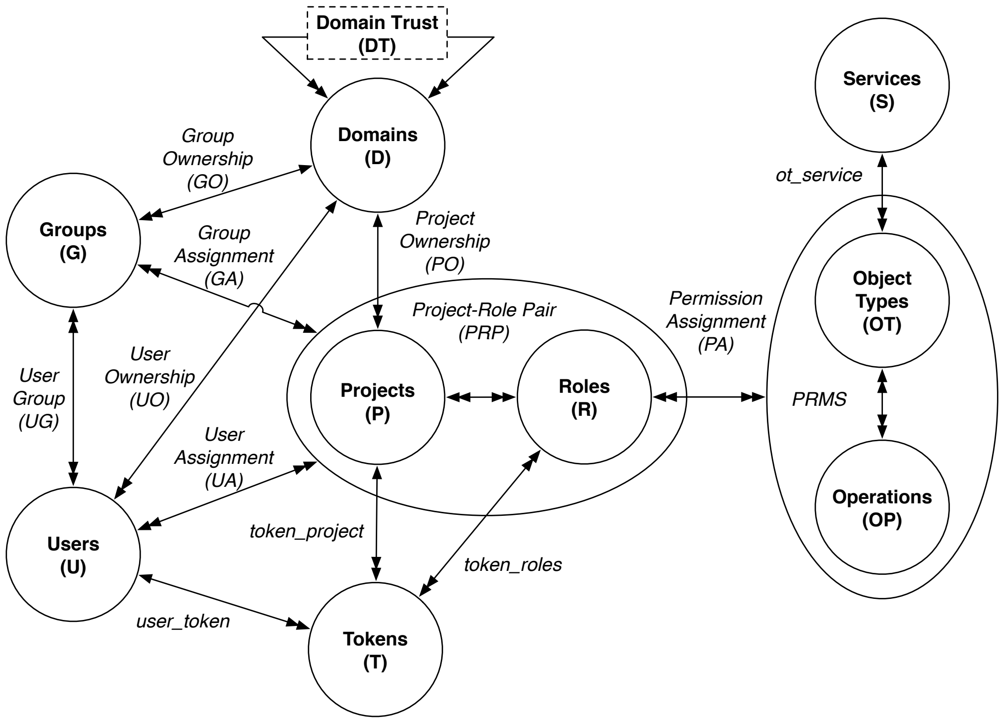


Figure 5.2: Core OpenStack Access Control (OSAC) model with domain trust.

provided by the identity service. The Domain Trust (DT) relationship is shown in dashed lines since it is not currently part of OpenStack but is proposed as an extension in this dissertation.

Users and Groups. A user represents an individual who can authenticate and access cloud resources. In OpenStack, users are the only consumers of cloud resources. A group is simply a collection of users. Each user or group is owned by one and only one domain. Each group contains only users in its owning domain. Since groups share the nature of users, for convenience we understand "users" to mean "users or groups" in the rest of this dissertation.

Projects. A project is a scope and/or a container of cloud resources. A project manages multiple services and a service segregates its resources into multiple projects. Using a project and a service, we can locate a specific set of resources. For example, the compute service of *Sales.Production* project manages the virtual machine (VM) instances of production applications for the sales department. Each project is owned by one and only one domain.

Domains. A domain is an administrative boundary of users, groups and projects. Domains are mutually exclusive. Each user, group and project belongs to one and only one domain.

Roles. Roles are global names which are used to associate users with any of the projects. A user is assigned a role with respect to a project, in other words to a project-role pair.[3] Users can be authorized permissions only through roles. The functions of roles may vary drastically in different services depending on the nature of the service.

Services. A service represents a distributed cloud service. Since OpenStack and most of other cloud systems are designed following service-oriented architecture (SOA) model, cloud applications and resources are delivered to the customers as services. The core service types in OpenStack include compute, image, identity, volume and network.

Object Types and Operations.[4] An object type represents a kind of cloud resources such as VM or image. Each service may provide multiple object types. For example, within the network service, IP and port are different object types. An operation is an access method to the object types. General operations are create, read, update and delete (CRUD) interacting with object types. For example, a typical permission "Delete VM" is a combination of delete operation and VM object type. Note that in cloud environments we cannot specify a particular object in the policy since the objects are created "on-demand". Thus, the finest-grained access control unit is a collection of objects identified by a specific object type and a specific project.

As a role-based authorization model, the central part of OSAC is the assignments related to roles: user assignment (UA), group assignment (GA) and permission assignment (PA) as illustrated in Figure 5.2. Both groups and users are assigned to project-role pairs but permissions are assigned to roles. As a result, the permissions assigned to a role populates across all the projects. For example, if Dan is assigned a developer role in both $Sales.Development$ and $HR.Development$, the permissions available to Dan through the developer role in both projects are identical. This arrangement embodies the multi-tenant nature of cloud resources and provides great flexibility for the assignments as long as the definition of roles is consistent within each service. It is worth

[3]Users can also be assigned a domain-role pair. This is for administrative usage only and will be discussed in Section 5.2.

[4]For clarity, we introduce object types and operations as components of permissions to the OSAC model. There is no specification of these two concepts in the identity service API.

noting that user assignments and group assignments are managed centrally in the identity service which permission assignments are distributed into each service.

Tokens. A token represents a subject acting on behalf of a user. A token is issued by the identity service for an authenticated user and then validated by other services whenever the user requests cloud resource accesses. A token may be expired or revoked during its lifetime. The content of a token is encrypted with public key infrastructure (PKI) so that it cannot be altered during transportation. It reveals all the information needed to authorize the access including the accessing user, the target project[5], all the assigned roles in the project[6] and service catalogs. The function $user\_tokens$ returns the set of tokens that are associated with a user, the function $token\_project$ returns the target project and the function $token\_roles$ returns the roles assigned to the user in the target project. Typically a user is issued one token for each project. Thus, in a particular project, the permissions available to the user are the permissions assigned to the roles revealed in the correlated token.

We summarize the above in the following definition.

Definition 14. *Core OSAC model has the following components.*

- U*,* G*,* P*,* D*,* R*,* S*,* OT*,* OP *and* T *are finite sets of users, groups, projects, domains, roles, services, object types, operations and tokens respectively.*

- $user\_owner : U \to D$*, a function mapping a user to its owning domain. Equivalently viewed as a many-to-one relation* $UO \subseteq U \times D$*.*

- $group\_owner : G \to D$*, a function mapping a group to its owning domain. Equivalently viewed as a many-to-one relation* $GO \subseteq G \times D$*.*

- $project\_owner : P \to D$*, a function mapping a project to its owning domain. Equivalently viewed as a many-to-one relation* $PO \subseteq P \times D$*.*

[5]The accessing scope may be project, domain, or even unscoped. For ordinary accesses, a token is scoped to a project

[6]Currently, OpenStack does not support activating an arbitrary subset of roles assigned to a user in the project.

- $UG \subseteq U \times G$*, a many-to-many relation assigning users to groups where the user and group must be owned by the same domain.*

- $PRP = P \times R$*, the set of project-role pairs.*

- $PERMS = OT \times OP$*, the set of permissions.*

- $ot\_service : OT \rightarrow S$*, a function mapping an object type to its associated service.*

- $PA \subseteq PERMS \times R$*, a many-to-many permission to role assignment relation.*

- $UA \subseteq U \times PRP$*, a many-to-many user to project-role assignment relation.*

- $GA \subseteq G \times PRP$*, a many-to-many group to project-role assignment relation.*

- $user\_tokens : U \rightarrow 2^T$*, a function mapping a user to a set of tokens; correspondingly,* $token\_user : T \rightarrow U$*, mapping of a token to its owning user.*

- $token\_project : T \rightarrow P$*, a function mapping a token to its target project.*

- $token\_roles : T \rightarrow 2^R$*, a function mapping token to its set of roles. Formally,*
  $token\_roles(t) = \{r \in R | (token\_user(t), (token\_project(t), r)) \in UA\} \cup$
  $(\bigcup_{g \in user\_groups(token\_user(t)} \{r \in R | (g, (token\_project(t), r)) \in GA\})$.

- $avail\_token\_perms : T \rightarrow 2^{PERMS}$*, the permissions available to a user through a token, Formally,*
  $avail\_token\_perms(t) = \bigcup_{r \in token\_roles(t)} \{perm \in PERMS | (perms, r) \in PA\}$.

Role hierarchy (RH) is not supported in OSAC but it could be a reasonable extension for convenience. Depending on operation needs, the hierarchy relation may be added upon roles or to project-role pairs. Both approaches allow specification of role-hierarchy assignments in the centralized identity service while the former also supports distributed assignment since different service may build different structures of role hierarchy as needed. Consideration of these extensions is beyond the scope of this dissertation.

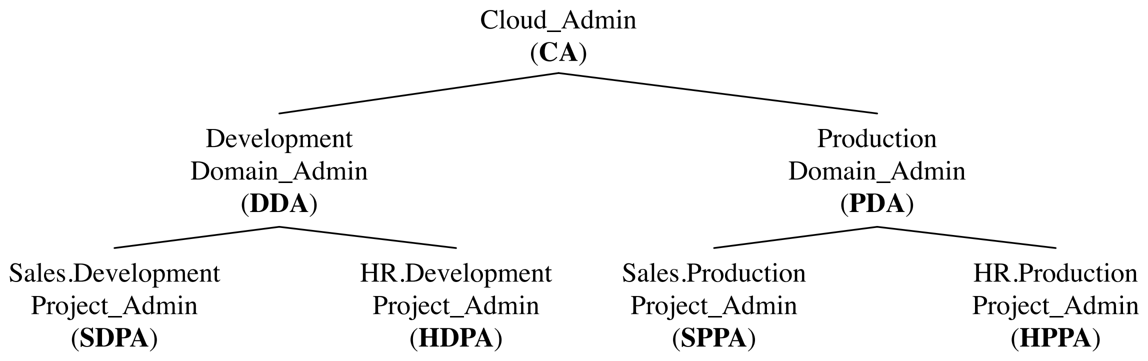


Figure 5.3: An example administrative role hierarchy.

Administrative OSAC Model

As described previously, the identity information of all the entities including services, domains, users, groups, projects and roles are stored and managed by the Keystone identity service in OpenStack, as are the assignments associating users and groups with roles in domains or projects. It is worth to note that the permission assignments are separately maintained by each cloud service provider in a policy file. The policy file for the identity service specifies the permissions to manage identities and assignments for administrator roles.

The administrative OSAC (AOSAC) model consists of three levels of administrative roles: $cloud\_admin$, $domain\_admin$ and $project\_admin$. As their names indicate, $cloud\_admin$ refers to top-level administrators with the CSP managing all the information in the identity service; $domain\_admin$ at the middle-level is able to conduct administrative tasks within the associated domain; and $project\_admin$ at the bottom-level take the responsibility of managing UA and GA assignments for the associated project. A user can only be assigned to $cloud\_admin$ role at the installation time of the cloud or by other users with the $cloud\_admin$ role afterwards. The $domain\_admin$ and $project\_admin$ roles are assigned to users by associating the users with the "admin" role in a specific domain or project respectively. Figure 5.3 illustrates an example administrative role hierarchy in AOSAC.

In the DevOps example described in Section 5.1, $Development\ domain\_admin$ (DDA) and

$Production$ $domain\_admin$ (PDA) roles are assigned to users owned by each domain respectively. A PDA can list and view users, groups and projects in $Production$. He or she can also assign roles, including the "admin" role, in a project of $Production$ to a user. A $Sales.Production$ $project\_admin$ ($SPPA$) can assign roles other than the "admin" role in $Sales.Production$ to a user. Note that a PDA or a $SPPA$ can assign $Dennis@Development$ to the "developer" role in $Sales.Production$. As a result, DevOps cross-domain accesses may be authorized. However, the administrative boundary of the two domains are intersected with each other. This may lead to unwanted authorization in cross-domain collaboration, such as the DevOps example.

5.3 Domain Trust Model

In order to achieve additional control for cross-domain accesses, we propose domain trust models integrating with the OSAC model. From the description in the previous sections, we observe that domains are introduced as administrative boundaries. Bridging domains using trust relations gives a controlled way to allow cross-boundary collaborations. For a user to have roles in a project, a proper trust relation needs to be established between the owning domains of the user and the project.

Domain Trust Relation

The definitions of trust relations vary in different application scenarios. In the field of access control, either explicit or implicit trust relation is essential to decentralized authorization [20]. Thus, in order to properly authorize cross-domain accesses, we have to specify what a trust relation means and how the trust relation interacts with the existing access control model.

Trust is a complicated concept and has been treated in different ways in the context of access control. The following is a list of characteristics related to domain trust relations. Figure 5.4 depicts the potential combinations.

Protocol (Two-party vs Federation). Two-party trust is established between two domains. A federation trust exists in an alliance or cooperative association in which a participant that is a

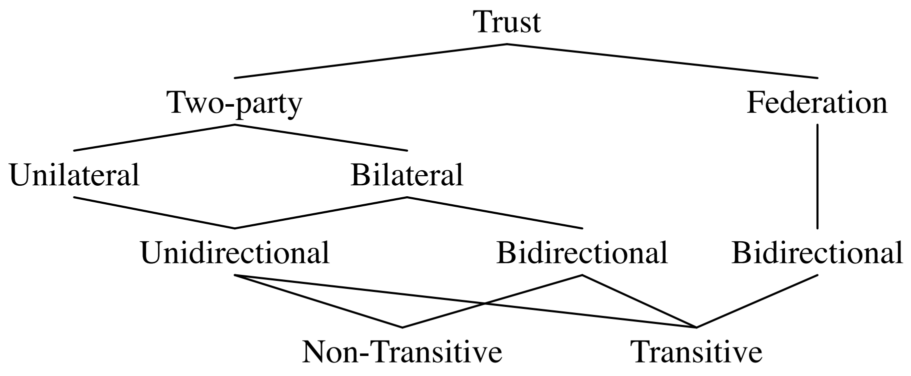


Figure 5.4: A tree structure showing characteristics of domain trust relation.

domain trusts all other participants and it is also true in return.

Initiation (Bilateral vs Unilateral). When the trustor creates a trust relation, if the trustee is required to confirm, then the trust relation is regarded as bilateral otherwise unilateral. It is worth to note that transferring a unilateral trust relation to a bilateral one is much easier than doing the reverse.

Direction (Bidirectional vs Unidirectional). A bidirectional trust relation requires the actions enabled through the trust relation are equally available for the trustor and the trustee. Conversely, a unidirectional trust, as the name refers, requires availability of the actions only on one side.

Transitivity (Transitive vs Non-transitive). For domain A, B and C, if A trusts B and B trusts C it is implied that A trusts C, then the trust relation is transitive. Otherwise, the trust relation is non-transitive.

In this dissertation, the domain trust relation is specified as a two-party, unilateral, unidirectional and non-transitive relation. Moreover, it is reflexive meaning each domain trusts itself. It is functionally defined as the following.

Definition 15. *If and only if Domain A trusts Domain B, also written as "$A \trianglelefteq B$", A or B can perform unidirectional cross-domain authorization.*

The actions enabled through a domain trust relation depend on the trust types defined as follows.

Definition 16. *Based on collaborative access control needs, the domain trust relation described in Definition 15 can be categorized into three useful types.*

- ***Type-**α, requires visibility of the trustee's user information for the trustor to assign trustee's users to roles in trustor's projects, written as "$\trianglelefteq_\alpha$".*
- ***Type-**β, requires the trustor to expose its user information for the trustee to assign trustor's users to roles in trustee's projects, written as "$\trianglelefteq_\beta$".*
- ***Type-**γ, requires the trustor to expose its project information for the trustee to assign trustee's users to roles in trustor's projects, written as "$\trianglelefteq_\gamma$".*

Note that the trust types defined in Definition 16 are consistent with those described in Section 3.3.3. Type-α Trust is used implicitly in current OpenStack since the trustor $domain\_admin$ and $project\_admin$ can see the users in all the domains and assign them to roles in the trustor's projects. Type-α Trust is only useful when user related information is not sensitive and available across domains by default. In contrast, Type-β Trust and Type-γ Trust protect user information as sensitive property of each domain. Both of them require dual control. In particular, the trustor manages the trust relation while the trustee manages cross-domain authorization. In this way, cross-domain accesses can be revoked by either end of the trust relation.

OSAC Domain Trust

Since we have specifically defined the domain trust relation above, integrating it with OSAC becomes straightforward. The formal definition of the OSAC Domain Trust (OSAC-DT) model follows.

Definition 17. *The OSAC-DT model extends the OSAC model in Definition 14 with the following modifications.*

- *$DT \subseteq D \times D$, a many-to-many trust relation on D, also written as "$\trianglelefteq$".*

- *UA is modified to require that $(u,(p,r)) \in UA$ only if*

  $project\_owner(p) \equiv user\_owner(u) \vee$

  $project\_owner(p) \trianglelefteq_\alpha user\_owner(u) \vee$

  $user\_owner(u) \trianglelefteq_\beta project\_owner(p) \vee$

  $project\_owner(p) \trianglelefteq_\gamma user\_owner(u)$.

- *GA is modified to require that $(g,(p,r)) \in GA$ only if*

  $project\_owner(p) \equiv group\_owner(g) \vee$

  $project\_owner(p) \trianglelefteq_\alpha group\_owner(g) \vee$

  $group\_owner(g) \trianglelefteq_\beta project\_owner(p) \vee$

  $project\_owner(p) \trianglelefteq_\gamma group\_owner(g)$.

The modification focuses on the effect of the domain trust relation introduced. Particularly, the project owner has to have proper trust relation with the user owner for UA and GA to take effect. The trust relations are checked during both authorization time and accessing time. The appropriate trust relations need to exist before the cross-domain assignment is issued. If a trust relation is revoked, then the correlated cross-domain assignments and accesses should be revoked either automatically or manually depending upon implementation.

OSAC-DT allows the three types of trust relations to coexist with each other. A specific cross-domain UA or GA is effective as long as the trust relation between the user or group and the project domains satisfy the condition described in Definition 17. In fact, combining Type-α and Type-β trusts we could achieve a bilateral trust relation. For example, only if both $Production \trianglelefteq_\alpha Development$ and $Development \trianglelefteq_\beta Production$ exists, then cross-domain authorization by $Production$ is enabled.

By introducing explicit domain trust relation, the following constraints may be enforced over cross-domain authorization.

Separation of Duties (SoD). Some of the collaborations among domains may have conflict of interests which should be addressed by additional constraint policy and lists of mutually exclusive domains.

Minimum Exposure. In collaboration, the over-exposure of user or project information increases security and privacy risks. An effective solution is limiting exposure of information based on each domain or each trust requirements.

Cardinality. A domain may limit the number of domains to be trusted. For example, some domains , such as $Production$, require high-level security and allow only one trusted domain at a time for temporary access if necessary.

The constraints listed above and a lot more are previously not available without domain trust relations.

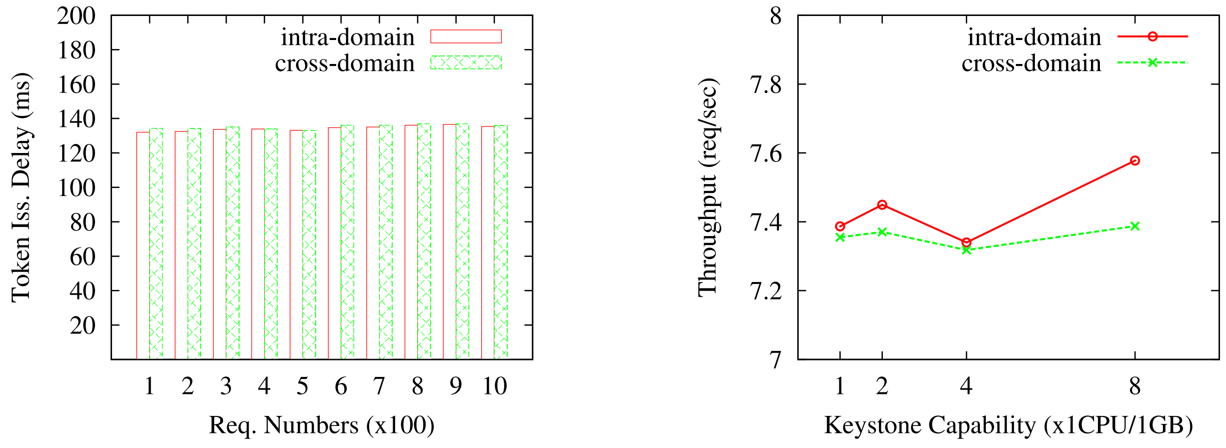
Domain Trust Administration

The administrative OSAC-DT (AOSAC-DT) model extends the AOSAC model by the administration of domain trust relations and their enabled actions. Since the trust relation is unilateral, only the $cloud\_admin$ and the $domain\_admin$ of the trustor have the permission to create and revoke a specific domain trust relation. The trust relation enables the $project\_admin$ and $domain\_admin$ of the trustor, in case of Type-α trust, or the trustee, in case of Type-β trust or Type-γ trust, to view the user or project information necessary for them to make cross-domain authorization.

5.4 Prototype and Evaluation in OpenStack

To further explore the feasibility of our OSAC-DT model, we implement a prototype system based on the Havana release of Keystone source code [6]. Furthermore, we conduct experiments on the prototype system in terms of performance and scalability. The results indicate that the integrated domain trust introduces minimum authorization overhead.

Implementation Overview The architecture of our prototype follows the Keystone design. The domain trust verification process intercepts the authentication process. Before Keystone issues the token for a requesting user, the domain trust relations stored in the MySQL database are checked. Only if the requesting user's owning domain is trusted by the target project's owning



(a) Performance results

(b) Scalability results

Figure 5.5: Performance and scalability evaluation results

domain, then the token issuing process can go through. Otherwise, an "unauthorized" response will be returned. For proof of concept purpose, we implement only Type-γ trust in the prototype system. It is straightforward to extend the implementation other types of domain trust relations discussed in Section 5.3 and similar evaluation results are predicted because the domain trust verification processes are similar.

Evalutation The implementation and experiments are conducted in experimental Devstack [2] deployments in a private cloud. The core OpenStack services, including Keystone, are running on a single VM. The requesting clients are from the same data center network of the private cloud. Since only Keystone code is modified, the experiments focus on evaluating the token issuing process including sequential processes of authentication, domain trust verification, token composition and network transmission, etc.

The experiments simulate sequential token requests for one hundred users and projects owned by ten independent domains. Each user is associated with both intra-domain and cross-domain assignments through ten different roles. As Figure 5.5a shows, the x-axis represents the requests per user and the y-axis indicates the latency between request time and response time from the client end, also known as the token issuing delay. Comparing the token issuing delay of intra-domain and cross-domain access requests, the domain trust verification process costs 0.96 ms on average or 0.7% performance overhead which is acceptable.

Figure 5.5b presents the results for scalability tests on our prototype system. The x-axis represents the capability of the VM running Devstack in the unit of “1CPU/1GB RAM”. The y-axis is the calculated throughput for the token issuing process. The plotted diagram shows that with ten requests per user, the throughput increase of the prototype system is proportional to the increase of the capability of Keystone servers from 1 unit to 8 units so that the system is scalable and adding the domain trust does not cause scalability problem.

Chapter 6: CONCLUSION AND FUTURE WORK

This chapter summarizes the contributions of this dissertation and the identified research directions for future studies of MTAC.

6.1 Summary

In this work, we propose a full stack of multi-tenant access control solutions for cloud services. The motivating example is the collaboration among multiple tenants in the daily business of enterprises. It is distinguished with the traditional multi-domain approaches by the characteristics of the emerging cloud environments. We use a top-down methodology in the research referring the PEI stack. Firstly, the policy layer models at the top are designed. Then, the enforcement layer architecture in the middle are developed to accommodate all the upper layer models. Finally, the prototype implementation of our models upon the architecture is based on a real-world cloud system.

In the policy layer, we give MTAC models both in role-based and attribute-based manners. The MTAS and MT-RBAC models are both role-based while the MT-ABAC model is attribute-based. Further, we summarize the generic types of trust relations and a unified framework of cross-tenant trust models which describes our basic approach in solving multi-tenant authorization problems.

In order to enforce our models, we develop a centralized authorization service architecture named MTAaaS. It has a centralized PDP and distributed PEPs for each tenant. The feasibility of MTAaaS is tested agains a prototype system using XACML in a cloud environment. The experiment results show that the performance and scalability of the prototype are acceptable.

Lastly, we integrate the MTAC models into OpenStack, a popular open-source cloud system. Keystone, as the centralized identity service of OpenStack, is modified to support domain trust which is consistent with our MTAC models. The experiment results against the modified Keystone service are also reported.

All in all, this work consists of research in all the three layers of multi-tenant access control for cloud services. It is believed to be valid and useful in contemporary cloud systems.

6.2 Future Work

Besides the completed work above, some potential directions as listed in the following which could be addressed in our future work.

1. The MT-ABAC model may be explored in administration, enforcement and implementation.
2. More and finer-grained trust models may be investigated. Trust negotiation and graded trust relations may be introduced.
3. Multi-Tenant Provenance-Based Access Control (MT-PBAC) and Multi-Tenant Risk-Adaptable Access Control (MT-RAdAC) could become other potential multi-tenant access control models compatible with the MTAaaS platform.
4. Last but not least, extending MTAaaS OpenStack API to support attribute-based MTAC models.

BIBLIOGRAPHY

[1] DevOps. http://en.wikipedia.org/wiki/DevOps.

[2] Devstack. http://www.devstack.org.

[3] Joyent SmartOS. http://smartos.org/.

[4] Microsoft windows active directory. http://en.wikipedia.org/wiki/Active_Directory.

[5] OpenStack. http://www.openstack.org/.

[6] OpenStack Havana Release. http://www.openstack.org/software/havana.

[7] Openstack identity service API v3 (STABLE). http://developer.openstack.org/api-ref-identity-v3.html.

[8] Trust (social sciences). http://en.wikipedia.org/wiki/Trust_ (social_sciences).

[9] Walkthrough: Cross-account API access using IAM roles. http://docs.aws.amazon.com/IAM/latest/UserGuide/cross-acct-access-walkthrough.html.

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

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

[12] OASIS eXtensible Access Control Markup Language (XACML) v2.0 specification set. http://www.oasis-open.org/committees/xacml/, 2005.

[13] Mohammad A. Al-Kahtani and Ravi S. Sandhu. A model for attribute-based user-role assignment. In *Proceedings of the 18th Annual Conference on Computer Security Applications (ACSAC)*, pages 353–362, 2002.

[14] R. Alfieri, R. Cecchini, V. Ciaschini, Luca dell'Agnello, Á Frohner, K. Lőrentey, and F. Spataro. From gridmap-file to VOMS: managing authorization in a grid environment. *Future Gener. Comput. Syst.*, 21(4):549–558, 2005.

[15] A. Almutairi, M. Sarfraz, S. Basalamah, W.G. Aref, and A. Ghafoor. A distributed access control architecture for cloud computing. *IEEE Software*, 29(2):36–44, 2012.

[16] Nathalie Baracaldo, Amirreza Masoumzadeh, and James Joshi. A secure, constraint-aware role-based access control interoperation framework. In *Proceedings of the 5th International Conference on Network and System Security (NSS)*, pages 200–207. IEEE, 2011.

[17] E. Barka and R. Sandhu. Framework for role-based delegation models. In *Proceedings of the 16th Annual Conference on Computer Security Applications (ACSAC)*, pages 168 –176, 12 2000.

[18] Rafae Bhatti, Elisa Bertino, and Arif Ghafoor. X-FEDERATE: A policy engineering framework for federated access management. *IEEE Transactions on Software and Engineering*, 32:330–346, 2006.

[19] Rafae Bhatti, Elisa Bertino, and Arif Ghafoor. An integrated approach to federated identity and privilege management in open systems. *CACM*, 50(2):81–87, 2007.

[20] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE, 1996.

[21] D.F.C. Brewer and M.J. Nash. The chinese wall security policy. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 206–214, 1989.

[22] Jose M. Alcaraz Calero, Nigel Edwards, Johannes Kirschnick, Lawrence Wilcock, and Mike Wray. Toward a multi-tenancy authorization system for cloud services. *IEEE Security & Privacy*, Nov/Dec 2010:48–55, 2010.

[23] David W. Chadwick and Alexander Otenko. The PERMIS X.509 role based privilege management infrastructure. In *Future Generation Computer Systems*, pages 135–140. Elsevier, 2002.

[24] DavidW. Chadwick. Federated identity management. In *Foundations of Security Analysis and Design V*, volume 5705 of *Lecture Notes in Computer Science*, pages 96–120. Springer Berlin Heidelberg, 2009.

[25] Frederick Chong, Gianpaolo Carraro, and Roger Wolter. Multi-tenant data architecture. http://msdn.microsoft.com/en-us/library/aa479086.aspx, 2006.

[26] Raul F Chong. Designing a database for multi-tenancy on the cloud. http://www.ibm.com/developerworks/data/library/techarticle/m-1201dbdesigncloud/index.h tml, 2012.

[27] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, August 2001.

[28] I. Foster, Yong Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop (GCE)*, pages 1–10, 2008.

[29] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti. dRBAC: distributed role-based access control for dynamic coalition environments. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, pages 411–420, 2002.

[30] Samy Gerges, Sherif Khattab, Hesham Hassan, and Fatma Omara. Scalable multi-tenant authorization in highly collaborative cloud applications. *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, 2(2):106–115, 2013.

[31] Li Gong and Xiaolei Qian. Computational issues in secure interoperation. *IEEE Transactions on Software Engineering*, 22(1):43–52, 1996.

[32] Michael T. Goodrich, Roberto Tamassia, and Danfeng (Daphne) Yao. Notarized federated ID management and authentication. *J. Comput. Secur.*, 16(4):399–418, 2008.

[33] Vincent C. Hu, David Ferraiolo, Rick Kuhn, Arthur R. Friedman, Alan J. Lang, Margaret M. Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (ABAC) definition and considerations (draft). NIST special publication 800-162, April 2013.

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

[35] Xin Jin. Attribute-based access control models and implementation in cloud infrastructure as a service. Spring 2014.

[36] Xin Jin, Ram Krishnan, and Ravi Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. In *Data and Applications Security and Privacy XXVI*, pages 41–55. Springer, 2012.

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

[38] D Richard Kuhn, Edward J Coyne, and Timothy R Weil. Adding attributes to role-based access control. *Computer*, 43(6):79–81, 2010.

[39] Anil Kurmus, Moitrayee Gupta, Roman Pletka, Christian Cachin, and Robert Haas. A comparison of secure multi-tenancy architectures for filesystem storage clouds. In *Middleware*, pages 471–490, 2011.

[40] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

[41] Qi Li, Xinwen Zhang, Mingwei Xu, and Jianping Wu. Towards secure dynamic collaborations with Group-Based RBAC model. *Computers & Security*, 28(5):260–275, 2009.

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

[43] Peter Mell and Timothy Grance. The NIST definition of cloud computing. Special Publication 800-145, 2011.

[44] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 50–59. IEEE, 2002.

[45] Juan M Marin Perez, Jorge Bernal Bernabe, Jose M Alcaraz Calero, Felix J Garcia Clemente, Gregorio Martinez Perez, and Antonio F Gomez Skarmeta. Taxonomy of trust relationships in authorization domains for cloud computing. *The Journal of Supercomputing*, pages 1–25, 2014.

[46] Christy Pettey and Rob van der Meulen. Gartner outlines five cloud computing trends that will affect cloud strategy through 2015. Press Release, 2012.

[47] Ravi Sandhu. The PEI framework for application-centric security. In *Proceedings of the 1st International Workshop on Security and Communication Networks (IWSCN)*, pages 1–6, 2009.

[48] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[49] Basit Shafiq, James BD Joshi, Elisa Bertino, and Arif Ghafoor. Secure interoperation in a multidomain environment employing RBAC policies. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1557–1577, 2005.

[50] Mohamed Shehab, Elisa Bertino, and Arif Ghafoor. SERAT: SEcure role mApping technique for decentralized secure interoperability. In *Proceedings of the 10th ACM symposium on Access control models and technologies (SACMAT)*, pages 159–167, 2005.

[51] Hassan Takabi and James B. D. Joshi. Semantic-based policy management for cloud computing environments. *International Journal of Cloud Computing*, 1(2):119–144, 01 2012.

[52] Bo Tang, Qi Li, and Ravi Sandhu. A multi-tenant RBAC model for collaborative cloud services. In *Proceedings of the 11th IEEE Conference on Privacy, Security and Trust (PST)*, 2013.

[53] Bo Tang and Ravi Sandhu. Cross-tenant trust models in cloud computing. In *Proceedings of the 14th IEEE Conference on Information Reuse and Integration (IRI)*, 2013.

[54] Bo Tang and Ravi Sandhu. Extending openstack access control with domain trust. In *Proceedings of the 8th International Conference on Network and System Security (NSS)*, 2014.

[55] Bo Tang, Ravi Sandhu, and Qi Li. Multi-tenancy authorization models for collaborative cloud services. In *Proceedings of the 14th International Conference on Collaboration Technologies and Systems (CTS)*, 2013.

[56] William Tolone, Gail-Joon Ahn, Tanusree Pai, and Seng-Phil Hong. Access control in collaborative systems. *ACM Computing Surveys (CSUR)*, 37(1):29–41, 2005.

[57] E. Yuan and J. Tong. Attributed based access control (ABAC) for web services. In *Proceedings of the IEEE International Conference on Web Services (ICWS).*, pages 561–569, 2005.

[58] Xinwen Zhang, Sejong Oh, and Ravi S. Sandhu. PBDM: a flexible delegation model in RBAC. In *Proceedings of the 8th ACM symposium on Access control models and technologies (SACMAT)*, pages 149–157. ACM, 2003.

[59] Zhixiong Zhang, Xinwen Zhang, and Ravi Sandhu. ROBAC: Scalable role and organization based access control models. In *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 1–9. IEEE, 2006.

VITA

Bo Tang was born in Wuhan, Hubei, China. After completing his schoolwork at Wuhan No.11 High School in 2003, Bo entered Huazhong University of Science and Technology in Wuhan. He received both a Bachelor of Engineering in information security and a Bachelor of Arts in English from Huazhong University of Science and Technology in June 2007. During the following three years, he was employed as an information security engineer at Ping An Insurance Group of China in Shenzhen, Guangdong, China. In August 2010, Bo entered the Graduate School of The University of Texas at San Antonio. He earned a Master of Science in computer science with a concentration of computer and information security from The University of Texas at San Antonio in May 2014.