AUTHORIZATION FEDERATION IN MULTI-TENANT MULTI-CLOUD IAAS

APPROVED BY SUPERVISING COMMITTEE:

_____

Ravi Sandhu, Ph.D., Chair

_____

Ram Krishnan, Ph.D., Co-Chair

_____

Gregory White, Ph.D.

_____

Matt Gibson, Ph.D.

_____

Palden Lama, Ph.D.

DEDICATION

This dissertation is dedicated to my parents and my family, who patiently supported me all the way. I must also thank all my friends.

AUTHORIZATION FEDERATION IN MULTI-TENANT MULTI-CLOUD IAAS

by

NAVID PUSTCHI, 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
May 2016

ACKNOWLEDGEMENTS

First, I would like to thank my advisors Prof. Ravi Sandhu and Prof. Ram Krishnan for their professional guidance. Without their help, this work could not be accomplished. They have provided guidance on how to do research and also provided great research topics which motivated me to grow and learn faster. They taught me to learn not only how to find and solve practical technical problems but also how to be a better person in life. The emphasize on contributing to real world problems than just writing academic papers is one of the most important takeaways from Ph.D life.
Second, I would like to express gratitude to my other committee members Prof. Gregory White, Prof. Matt Gibson, and Prof. Palden Lama for their valuable comments and suggestions.
Third, I would like to thank my colleagues and friends in the lab. We learned the tools for implementation together and they've provided great help when I got stuck in programming issues. Especially, I want to thank Farhan Patwa, director of our lab. Without him, I could not finish the implementation project, which is a crucial part of my dissertation. During this period, I gained industry level experience from which further benefited me in job hunting.

May 2016

AUTHORIZATION FEDERATION IN MULTI-TENANT MULTI-CLOUD IAAS

Navid Pustchi, Ph.D.
The University of Texas at San Antonio, 2016

Supervising Professor: Ravi Sandhu, Ph.D.

Cloud computing significance has been proven in the marketplace and well documented in the literature. A major concern in adopting cloud Infrastructure-as-a-service (IaaS) is federation, where tenants engage in collaborative tasks requiring resources to be shared across tenant boundaries. Federation is a critical impediment to private, public, and hybrid cloud deployments today. The federated cloud model is a significant shift towards democratization in the cloud market. It enables businesses using local cloud providers to connect with customers, partners and employees anywhere in the world. In this context, cloud service providers (CSP) use multi-tenancy to consolidate economic utility of shared infrastructure by isolating users' data into tenants. Tenants are isolated containers owning resources such as users, storage objects, and virtual machines in the cloud. While tenant isolation is desirable, it hinders federation in cloud platforms.

Role-based access control (RBAC) has been widely accepted and applied in practice for over two decades. The majority of current cloud IaaS platforms adopt some variation of RBAC. It has been considerably investigated in terms of multi-tenancy, federation, policy integration, etc. However, to cover RBAC limitations, there has been considerable recent interest towards attribute-based and attribute integration to role-based models. Attribute-based access control (ABAC) also has been researched on various aspects such as policy languages and multi-tenancy. In order to effectively provide cloud computing federation with cloud intrinsic characteristics such as multi-tenancy, virtualization, and service oriented architecture (SOA) fine-grained cloud oriented access control models are required.

In this dissertation, we propose a set of access control models to enable federation in the cloud IaaS platform. Our contributions are categorized into two federation models, Peer-to-Peer model where trust is established between two tenants and Circle-of-Trust model where a group of tenants

adhere to agreed policies and interfaces to collaborate. In Peer-to-Peer federation, role-based and attribute-based models are proposed to enable cross-tenant access. We extend existing multi-tenant approaches into multi-cloud role-based access control model providing cross-cloud user assignments. Moreover, a novel attribute-based access control model providing Peer-to-Peer federation between tenants in a cloud IaaS, as well as more generally, is proposed. Our approach allows cross-tenant attribute assignment across tenants. Particularly, tenant-trust authorizes a trustee tenant to assign its attributes to users from a trustor tenant, enabling access to the trustee tenant's resources.

In Circle-of-Trust federation, we propose a suite of multi-tenant role-based, role-centric, and tenant-trust models in the context of homogeneous and heterogeneous circles. In a homogeneous circle with uniform tenant types, role-based approach allows tenants to equally assert cross-tenant user assignments. In role-centric attribute-based model, attributes are added to differentiate tenants in heterogeneous circles with non-uniform tenant types. Attributes are used to limit user-role assignments with respect to tenant types. Tenant-trust model provides user-role assignment in homogeneous and heterogeneous circles enabling federation in the circle. Particularly, it specifies user-role assignments with respect to rules and policies in the circle.

As a proof of concept, we demonstrate the feasibility of the proposed multi-tenant multi-cloud access control model by integrating into an open-source cloud IaaS platform. Particularly, OpenStack identity service is extended in an OpenStack to OpenStack federation, providing user-role assignments across distinct domains across different OpenStack clouds. Our implementations have minimal impact on administration and no impact on operation performance in OpenStack.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: INTRODUCTION

Cloud computing has revolutionized the way that IT resource are available to organizations [39]. It is altering the way software is developed, deployed, adopted, and paid for. Cloud computing brings on-demand devices with unlimited pool of computing power to mainstream. Its advantages are beyond just the drop in costs. It changes the way enterprises invest in their computing, storage, security, and applications.

A significant benefit of cloud computing is the elasticity and dynamicity it provides for cloud consumers in addition to advantages such as security, disaster recovery, etc. Cloud computing benefits over traditional computing is favored in tasks where demand for services varies in time, is unknown in advance, and demand for processing can be distributed over resources [2]. Stated characteristics by NIST [40] for cloud computing such as *on-demand self-service*, *resource pooling*, and *rapid elasticity* also emphasize the flexibility as an essential feature of clouds. Federation mechanisms are essential to further enhance flexibility and dynamicity of cloud platforms' service delivery. Federation binds distinct cloud platforms with trust relationships to share resources and services.

Moreover, cloud computing characteristics such as *infinite computing resources on demand*, *no up-front commitment*, and *pay-per-use* makes it ideal to fill organizations' IT portfolio with deploying their resources in the cloud [3]. Enterprise work-flow intrinsically mandates collaboration across its tenant boundaries as well as with associated organizations' tenants. A major challenge in cloud adoption is fine-grained collaboration models in cloud platforms.

Cloud service providers utilize multi-tenancy to share underlying physical infrastructure within tenants in cloud IaaS. Cloud providers segregate the resources and customer's data into tenants to protect data privacy and integrity. In cloud, tenants are isolated containers with tenant-specific virtual computing environments where each tenant corresponds to an organization, a department of an organization, or an individual who uses cloud services. In this scenario, each tenant is considered as a cloud customer with resources whose integrity and privacy must be protected. Multi-tenancy

adoption signifies necessity of collaboration between cloud consumers' tenants across a single-cloud or a multi-cloud environment.

Multi-tenancy requires collaboration to dynamically enable access to resources across tenants in a single-cloud IaaS [56] or in multi-cloud federation where tenants are located across distinct cloud platforms [4]. Multi-cloud collaboration paradigm allows cloud consumers to avoid vendor lock-in as well as enables providers to avoid limitation of restricted amount of infrastructure in cloud IaaS [55].

Currently cloud platforms such as Amazon AWS [1], OpenStack [48], and Microsoft Azure [41] offer federation APIs to enable collaboration between tenants. Tenant concept in AWS is represented by an account, in OpenStack by a domain, and in Azure by Active Directory tenant. Multi-tenant multi-cloud models proposed in this dissertation extend current federation APIs in cloud platforms such as OpenStack to role-based [54] and attribute-based [28] access control models.

1.1 Motivation

At the dawn of cloud systems, the multi-tenancy concern was resource segregation, whereas recent enterprise cloud adoption has raised the issue of multi-tenancy resource sharing. Collaboration in cloud IaaS is generally categorized into cloud bursting and cross-tenant access [63] scenarios.

Cloud bursting scenarios allows private clouds to establish partnership with a public cloud to utilize infrastructure resources [43]. The drive for multi-tenant collaboration arises from at least two distinct directions. First, a large organization may utilize multiple tenants for security and reliability, where each tenant represents a department. Such tenants are located in a public cloud or across public and private clouds. For example, an organization's financial department processes sensitive financial data while its marketing department publishes open information to the public. For privacy and integrity measures sensitive data must not be co-located with public data in a tenant. Distinct tenants with different security measures are required but yet may need controlled collaboration. Second, distinct enterprises may have collaborative tasks across their corresponding tenants. Collaboration between enterprises requires mechanisms to establish the collaboration and

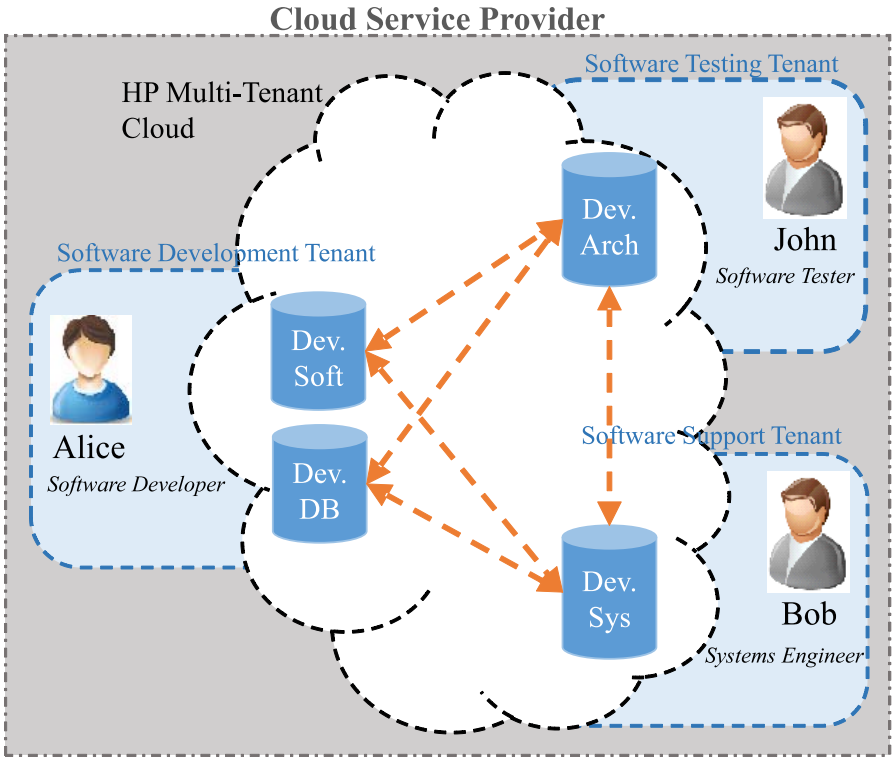


Figure 1.1: A Multi-Tenant Collaboration Example

at task completion to disband it.

To motivate the problem, consider the example illustrated in Figure 1.1, which depicts an organization with multiple tenants in a cloud service provider. We use HP as an organization with multiple locations and departments. In such organizations it is not feasible to locate all data and users into one tenant due to different security and reliability levels required as well as organizational structure. Creating user accounts across each collaborating tenant is impractical, whereas supporting access to shared resources is much more practical.

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

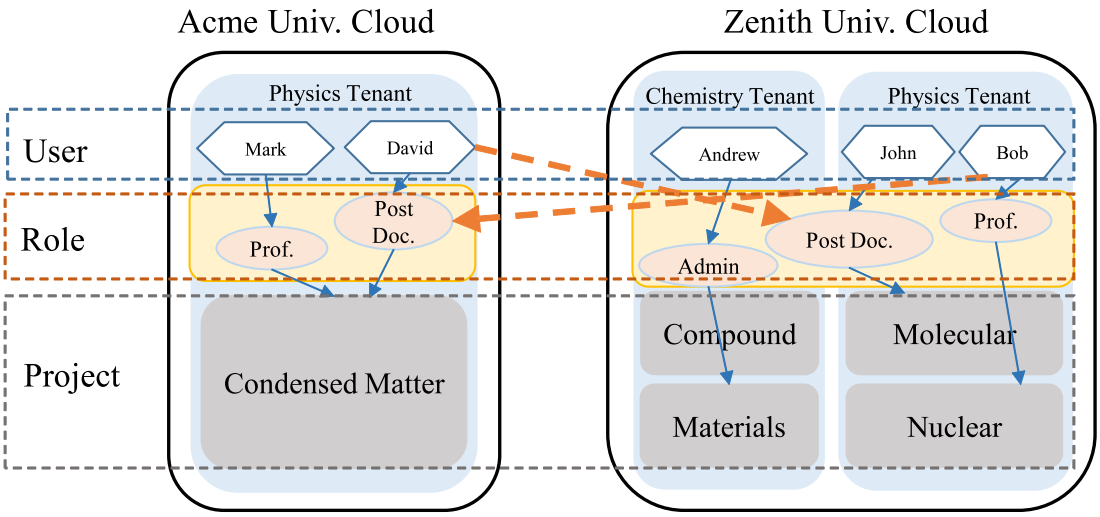


Figure 1.2: A Multi-Cloud Collaboration Example

Another collaboration example occurs when multiple organizations such as an inter-university research community collaborate to share data and processing power within a trusted community of researchers, illustrated in Figure 1.2. This Figure is based on the European Organization for Nuclear Research (CERN) which currently has 110 petabytes od data and 50 petabytes are added each year. The amount of data stored in participating institutes is so large that transmitting data to perform analysis is not practical. Moreover adding accounts for all the participating institutes' users in each individual cloud is also impractical.

Collaboration across institutes is achieved via an inter-university research community called CERN. We have two CERN member universities Acme and Zenith running OpenStack as their cloud platform. Bob is a professor in physics tenant in Zenith. For Bob to properly perform his analysis he should have access to Acme Cloud's project Condensed Matter. There should be cross-cloud access which enables Bob to perform his analysis. This can allow Bob to create a virtual machine (VM) in Acme cloud's Condense Matter project and perform analysis. Meanwhile David a postdoc in Acme Cloud requests to access Molecular tenant in Zenith Cloud This example is a typical use case for collaboration among multiple cloud providers. There are, similar use cases such as an organization which has resources distributed across multiple cloud service providers for certain security reasons and wishes to merge the administrative controls over all resources

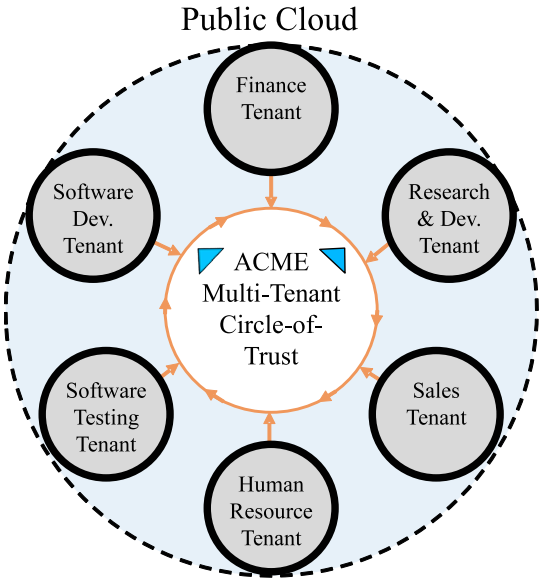


Figure 1.3: ACME Corporation Multi-Tenant Circle-of-Trust.

while each cloud still has separate administration. By enabling cross-cloud access we achieve the following benefits.

- We eliminate the need to provision users in every collaborating organization.
- Inter-cloud and intra-cloud assignments are differentiated and administered separately.
- Each participating organization has some degree of control over organizations' relationship.

Current federation APIs provide peer-to-peer collaboration where trust is established between a pair of tenants. Peer-to-peer trust is generally more appropriate when tenants' collaborative specifications cannot be extended to additional tenants. It is preferable due to limited trust, and presumably enhanced security. Recently, considerations on collaboration within a group of tenants has been considered in different contexts which is denoted as circle-of-trust. In a circle-of-trust, common policies and rules applies to participating partners.

To better clarify circle-of-trust concept, consider the example in Figure 1.3 where ACME, a multinational technology corporation aims to implement its enterprise requirements with cloud services.

ACME migrates its IT infrastructure to a public cloud service provider where each tenant represents a department. ACME utilizes multiple tenants to satisfy distinct security levels required for each department. For example, Finance Dept. resources should not co-locate in the same tenant with Research & Development Dept., as Finance Dept. retains sensitive data. Furthermore, ACME organizational structure demands collaboration between its departments which is thereby required in its cloud adoption. To this end, ACME establishes a circle-of-trust among its tenants in the cloud and starts adding its tenants to the circle. For instance a new tenant created as Sales tenant in ACME, requests to join the circle. Adding additional tenants requires all ACME circle members to agree on trusting the new Sales tenant. When Sales tenant joins the circle, it trusts members assertions and its assertions are likewise trusted by other ACME circle members.

1.2 Problem Statement

Role-based access control [17, 54] (RBAC) has been the dominant access-control paradigm for over two decades. While RBAC initially designed for a single organization [62], through many extensions proposed in the past decade it has been extended to encompass collaboration [59, 61]. The majority of RBAC extensions are not applicable to cloud IaaS directly. They enable collaboration to some extent but in terms of trust management administration and agility, limit the cloud IaaS features. Nevertheless, various limitations of RBAC have been recognized over this period and increasingly there is a push to move towards attribute-based access control [27, 28, 53] in general. ABAC advantages over RBAC specifically in cloud computing have been discussed in the literature [14]. Although considerable research has been devoted to attribute-based access control in the past decade, rather less attention has been paid to multi-tenancy and collaboration in ABAC.

Effective collaboration in multi-tenant cloud IaaS platforms requires generic models specifically built to acknowledge cloud characteristics. Current cloud platforms support partial federation mechanisms, they realize access control models in different manner and treat roles and attributes variously. For instance, current cloud platforms such as OpenStack implements RBAC with different interpretation of role and permissions from the standard RBAC. Policies heterogeneity in

cloud platforms make inter-operation across tenants more complicated. More specifically different realization of roles and attributes create semantic mistmatch for collaboration across tenants.

Fine-grained cloud collaboration models must provide a clear notion of trust. Establishing trust across tenants in cloud IaaS, administering trust relationships, and how trust affects each cloud's resources privacy and integrity are issues raised in cloud collaboration. Current cloud providers support basic trust across domains in OpenStack or accounts in AWS, however trust adminstration and extended collaboration models such as Circle-of-trust trust are not supported.

In Peer-to-Peer collaboration model, trust is established between a pair of tenants and upon type of trust relationship each tenant may authorized to assign users to resources across trusted tenant. Besides two tenants, collaboration can also be established between a set of tenants where tenants adhere to a common set of policies, trust relations and collaboration interfaces within a circle. We denote this collaboration model as a circle-of-trust. Current cloud platforms support Peer-to-Peer partially from account delegation in AWS to OpenStack cross-domain user assignments which is limited to simple Peer-to-Peer trust. Currently, cloud platforms are not cultivated to support various collaboration models such as Circle-of-Trust.

While multi-tenancy architecture brings economical and infrastructure utilization to cloud computing, segregating users and resources into tenants diminishes inter-operation efforts. In this architecture, each tenant has its own set of roles and attributes which tighten collaboration. Delegating appropriate access rights to users in cross-tenant access is crucial to overall cloud IaaS integrity and privacy. In a multi-tenant environment, access control administrative model must administer intra-cloud policies distinguishing inter-cloud policies. To that end, it is critical to manage authorization derived from trusting a tenant or trusted by a tenant.

Also policy decision points must be responsive enough where a large collection of policies controlling a shared resource in multiple tenants. The number of policies involved in access to cross-tenant resources affects the performance and results in policy inefficiency as redundancy and verbosity. If two policies matches same access request considered redundant. Policy integration can merge similar policies from multiple origins. Resolving the policy verbosity during composi-

tion leads to smaller policy size. Participating clouds must have authorization to specify constraints on policies to better control the multi-cloud relation. In a multi-cloud fashion policy enforcement points (PEPs) are distributed across clouds. PDPs are hosted in each cloud service and it is required that communication between services exist to retrieve necessary data for policy decision points.

Multi-cloud and distributed systems share similarities as both promise utility computing. Collaboration in distributed systems have been extensively researched for over two decades. However, cloud computing offers virtualization through hypervisor technology such as virtual machines, dynamically provisioned resources on-demand as a service which is accessible via Web service technologies such as SOAP and REST. Further, cloud computing offers services segregated to tenants where users and resources are owned by tenants. To this end, different characteristics of cloud computing accordingly makes prior role-based and attribute-based collaboration access control models impractical to apply for cloud collaboration directly. With this outline, we state our problem statement as following:

Current access control models provided by cloud platforms are not sufficient to cultivate effective peer-to-peer and circle-of-trust collaboration between tenants in a cloud or across multiple cloud platforms. Prior role-based and attribute-based access control models in distributed systems are not effectively applicable to cloud IaaS.

1.3 Scope and Assumptions

In this dissertation, we scope our contributions to Infrastructure-as-a-Service, homogeneous cloud platforms, Peer-to-Peer and Circle-of-Trust, and authorization federation. We identify collaboration in cloud computing upon characteristics such as service levels, cloud platforms, trust, and entity coupling illustrated in Figure 1.4.

Cloud computing offer software, platform, and infrastructure-as-a service (SaaS, PaaS, and IaaS) levels. Collaboration in cloud occurs between each service layer where in IaaS, services offered are homogeneous since provided service are generally computation, storage, networks, etc. However, in SaaS and PaaS services can be of heterogeneous type, such as Google account which

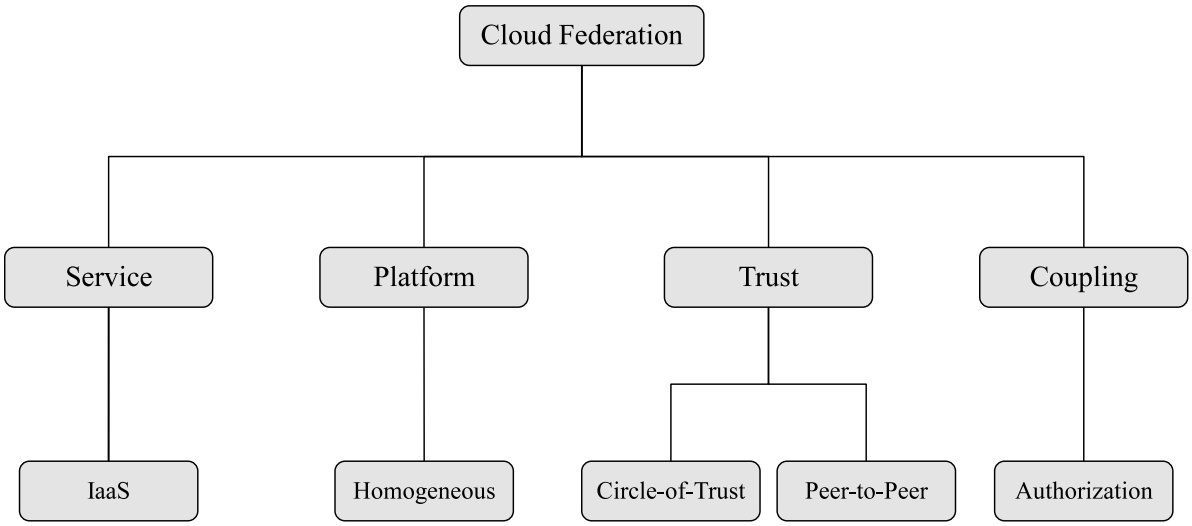


Figure 1.4: Scope of Contribution in Cloud Federation.

is a federation of heterogeneous services from Email and storage to mobile payments. In service levels, we focus on IaaS cloud platforms. Further, cloud platforms involved in collaboration based on deployment model form homogeneous and heterogeneous cloud federation. Access control models proposed are within homogeneous platform cloud federation.

In collaboration, trust established between cloud platforms (or tenants) define collaboration model enabled. Two distinct types of trust aries commonly in cloud collaboration scenarios, Peer-to-Peer and Circle-of-Trust. In this dissertation, our access control models trust scope are Peer-to-Peer and Circle-of-trust.

Moreover, in a cloud federation users accessing resources which are not located in their home tenant, therefore users must be authenticated prior to granting access. Authentication federation in coupling of cloud platforms which is concerned with authenticating users in clouds other than their home cloud (where they are initially authenticated). Further, authorization federation focus on authenticated users' access rights in resource provider cloud (or tenant). With this outline, we scope our contribution depicted in Figure 1.4 as authorization federation in Peer-to-Peer and Circle-of-Trust within homogeneous IaaS multi-tenant cloud platforms. This research is conducted with the following assumptions.

Standardized Cloud APIs. In order to develop collaborative access control models, cloud

platforms must have fundamental identity service functionalities as authentication, authorization, etc.

Federation APIs. We assume current cloud platforms have fundamental federation functionalities to generate and consume attributes such, for instance, SAML assertions in OpenStack federation API.

Authenticated Users. In this research, we assume users requesting access are already authenticated properly to better focus on authorization federation.

Platform homogeneity. We assume cloud platforms are homogeneous. We scope our contribution to homogeneous platforms to better focus on collaborative access control models.

Tenant Trust. For simplicity we define trust between tenants with two model of collaboration Peer-to-Peer and Circle-of-trust as follows. *Peer-to-Peer Trust.* Trust between two tenants are considered as unidirectional, unilateral and non transitive trust relations (see section 3.4). *Circle-of-Trust.* Trust between tenants in a circle-of-trust is considered as multilateral, bidirectional, transitive relation in homogeneous circles, and multilateral, unidirectional, non-transitive relation in heterogeneous circles (see section 3.4).

1.4 Thesis

The central thesis of this dissertation is:

The problem of authorization federation in multi-tenant cloud IaaS can be partially solved by integrating multiple types of peer-to-peer and circle-of-trust relations between tenants in single-cloud and multi-cloud environments into role-based and attribute-based access control models.

1.5 Summary of Contributions

We summarize our contributions into *Peer-to-Peer policy*, *Circle-of-Trust policy*, and *implementation* in this dissertation.

Peer-to-Peer Policy. In Peer-to-Peer federation model, we define a multi-cloud multi-tenant

role-based access control model and a multi-tenant attribute-based access control model (MT-ABAC) for cloud IaaS. Multi-cloud role-based model extends tenant-trust [59] type α, β, and γ in addition to type δ to tenants across homogeneous multi-cloud IaaS platforms. In our model, trust is defined as tenant-trust, authorizing trustor or trustee tenant to make user-role assignments upon applied trust type (α, β, γ, and δ). MT-ABAC, provides collaboration in an attribute-based multi-tenant cloud IaaS. Our approach allows cross-tenant attribute assignment to provide access to shared resources across tenants. Particularly, tenant-trust authorizes a trustee tenant to assign its attributes to users from a trustor tenant in type-β, enabling access to the trustee tenant's resources. Additionally, type-α is defined to authorize trustor tenant to assign its attributes to users from a trustee tenant and type-γ authorizes trustee tenant to assign trustor tenants' attributes to its users. We also demonstrate that MT-ABAC can be configured to enforce MT-RBAC thus subsuming it as a special case. In general, tenant-trust in our Peer-to-Peer polices are defined as unilateral, unidirectional, and non-transitive relations.

Circle-of-Trust Policy. In Circle-of-Trust federation model, we elaborate a multi-tenant role-based access control model (MT-RBAC$_c$) and a multi-tenant role-centric attribute-based access control model (MT-RABAC$_c$). In a Circle-of-Trust, we identify two types of circle, homogeneous circle where entities are from uniform type and heterogeneous circle which is an association of non-uniform entities. To this end, multi-tenant roles-based model enables collaboration in homogeneous circles, allowing tenants to equally assert cross-tenant user assignment. In a homogeneous circle, member tenants trust each other where trust is defined as tenant-trust. We define two types of tenant trust in Circle-of-Trust, ϵ and ζ enabling user-role assignment in the circle. Particularly, type-ϵ tenant-trust authorizes user-owner tenants to assign their users to public-roles in the circle and type-ζ tenant-trust authorizes role-owner tenants to assign users in the circle to their public-roles. Further, in multi-tenant role-centric attribute-based model attributes are associated with user, object, and tenant components to distinguish user-assignments where tenants are differentiated with type in the heterogeneous circle.

Implementation. Presented models are implemented in OpenStack cloud platform as an open-

source cloud IaaS. OpenStack identity service federation APIs, support fundamental trust between cloud platforms in which extended with tenant-trust within a single-cloud and multi-cloud enabling cross-tenant user-assignments. The results represents that the extended tenant-trust implementation introduces minimum administration overhead and without any operation performance adjustment.

1.6 Organization of Thesis

Chapter 2 gives a literature review of the related works including cloud federation, role-based and attribute-based access control models, federation, and OpenStack architecture with federation extensions. In Chapter 3, we review our framework in terms of cloud federation, multi-cloud, our cloud federation framework, and tenant-trust following by scope of contributions. Chapter 4 presents Peer-to-Peer federation model with role-based tenant-trust in multi-cloud IaaS with OpenStack implementations. Administrative model in terms of establishing tenant-trust and user-role assignments is demonstrated as well. In Chapter 5, we consider multi-tenant attribute-based access control model in Peer-to-Peer federation model. Peer-to-Peer tenant-trust is established as attribute assignment across tenants. The model is formalized and further we demonstrate our models is capable of MT-RBAC as a special case. In Chapter 6, Circle-of-Trust federation in homogeneous and heterogeneous circles elaborated. Moreover, a role-based access control model in circle (MT-RBAC$_c$) is demonstrated enabling user-role assignments in homogeneous circles and a role-centric attribute-based models in Circle-of-Trust is presented. Chapter 7 concludes the dissertation and discusses the future work.

Chapter 2: BACKGROUND AND RELATED WORK

This chapter review prior relevant work on role-based and attribute-based access control models, federation, and current OpenStack identity service. We review cloud federation concepts in section 2.1. We summarize related prior research on role-based and attribute-based models in sections 2.2 and 2.3 respectively. Finally, current OpenStack cloud platform identity service and its federation extensions are reviewed in section 2.4.

2.1 Cloud Federation

Several computing paradigms such as cluster computing [12], grid computing [20], and now cloud computing have promised utility computing vision where users access services based on their need without knowledge of how services are delivered. Grid computing is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime [6].

Cloud computing shares similarity in utility vision with grid computing, however it is more oriented towards services. Cloud computing consists of virtualized resources which are dynamically provisioned and presented as a ubiquitous, on-demand, and unified computing resource with measured service [6, 40]. Cloud computing provides more service oriented architecture (SOA) as a unified computing resource with virtualized resources rather than more application oriented grid computing. The core phenomena of cloud computing is that users can access services independently without reference to the underlying hosting infrastructure.

Resource sharing among organizations is not a new concept, Virtual organizations [45], have been developed in the grid computing community since 2001 which is comparable to federation in cloud with similar concepts and characteristics. Federation for resources sharing has been researched extensively in grid computing and distributed systems with respect to distributed trust, resource allocation, multi-domain access, etc. For instance, Condor-G [21] enables users to access multi-domain resources as tough they all belong to a single domain. It presents a single system

view of multiple distributed resources including clusters of computers regardless of their domain which creates a global grid designed to run jobs across different administrative domains. Grid Federation [52] focuses on multi-cluster systems to couple these cluster resources as a part of one large grid in different scale of cluster, campus, and global grids.

Collaboration among clouds will require cloud characteristic such as multi-tenancy, service orientation, and ubiquitous accessibility which distinguishes it from federation models in grid, cluster and in general distributed systems. In the literature, collaboration in cloud has been referred in various fashion as cloud federation [35], inter-cloud [23], hybrid cloud [40], and multi-cloud [50]. Moreover, considering that cloud computing distinguishes service models as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) [40], collaboration in cloud may occur in any service model.

In [35], authors define cloud federation as comprising services from different service providers aggregated to a single pool where the federation of resource migration, resource redundancy and combination of complementary resources are provided by cloud federation mechanisms. Also, two types of federation called horizontal and vertical federation are recognized where horizontal federation occurs within one service model (IaaS, PaaS, or SaaS) and vertical federation spans multiple service models. Inter-cloud has been used throughout the literature denoting as a cloud federation term where in [23], authors describe inter-cloud as a model where performance and availability of services is the purpose of collaboration. It allows reassignment of resources and transfer of workload through inter-connection of different cloud service providers where each cloud, provides service level agreements and standard interfaces. Hybrid cloud encompasses two or more types of cloud deployments as public, community and private as defined by NIST [40]. It is composed of distinct cloud infrastructures connected with standardized or proprietary technology enabling data and application portability.

Multi-cloud has been defined in various forms in the literature. In [50] multi-cloud means usage of multiple and independent cloud platforms by a client or service. Two categories of multi-clouds are recognized in [23], service based and library based. In service based, a service is offering

brokerage between clouds on client side. In library based, set of APIs offer uniform approach to access resources and services, as well as provisioning of services and resources from multiple clouds. Another method of collaboration between multiple clouds is using proxies as mediators between applications across distinct clouds, sharing data and collaboration is discussed in [55].

Moreover, there are implemented systems that provide cloud federation services such as OpenNebula [47], Eucalyptus [26], Aneka Coordinator [51], CometCloud [13], OpenStack [48], AWS [1], and Azure [41]. OpenNebula, Eucalyptus, and OpenStack are open-source cloud platforms which all offer federation APIs for cloud IaaS. Aneka coordinator is a resource management tool in the Aneka enterprise cloud to communicate and share resources with other Aneka clouds. It is composed of services to interact with Aneka's cloud core services providing functional peer-to-peer scheduling, peer-to-peer execution, and load balancing among the distributed Aneka enterprise clouds. CometCloud is an autonomic computing engine that enables federation of clouds. Amazon AWS and Microsoft Azure are cloud platforms which both offer federation APIs.

These group of collaborative clouds form federations. A federation can be defined as an organizational structure where multiple organizations have set up collaborative agreements [16]. Each organization has a administration and domain coupled to other organizations by trust agreements. Federation can be specified by its specific characteristics in terms of service, platform, trust and coupling of organizations (see chapter 3).

In identity federation two concepts arises authentication federation and authorization federation. A handful of formal standards and protocols have been proposed for web services and SOA in forms of federation such as SAML [30], OpenID [46], ID-FF [9], Shibboleth [42], WS-Federation [22], and OAuth [24] where SAML, ID-FF, Shibboleth, WS-Federation, and OAuth support some type of attribute assertions regarding authorization federation.

Security Assertion Markup Language (SAML) provides a means for exchanging security information. SAML specification is general in the kinds of assertions that can be transferred including authentication and authorization assertions. A major feature of OpenID is its user-centric approach which denotes that users can choose identity provide they trust to authenticate. OpenID is mostly

an identity protocol and federation is enabled through extensions to allow attribute exchange. ID-FF is the Liberty Alliance identity Federation Framework, provided on SAML basis to enable identity federation through single sign-on, identity linkage, and session management. Shibboleth is an open-source project implementing identity federation based on SAML specifications. It provides single sign-on and attribute exchange framework used mostly in educational environments. It provides extended privacy functionalities allowing users' identity provider to control attributes asserted to other applications. WS-Federation as part of Web Services Security Framework defines mechanisms to allow different security domains to federate. Authorized access is granted to security principals whose identity and attributes are originated from another security domain. OAuth provides a simple way to verify the access level of a request for a web service. Its authorization protocols provide a mechanism for application users to delegate access to a third-party to work on behalf of the user (within the scope of authorization server and token delegation). While such mechanisms can be adopted in an authorization federation at the SaaS service level, they are not suitable for IaaS because determining access rights consist of user attributes in addition to service providers' concession. The basic concept of authentication federation is trusted relationship between identity providers (which can be a cloud service provider in our context) and service providers. Federation Identity (authentication) Management has been widely researched, providing solutions by enabling propagation of identity information to services located in different administrative domains [10, 11].

Cabarcos et al [7] proposed a generic extension for SAML to create identity federation relations between unknown parties dynamically with certificate based and reputation based trust decisions. Also Chadwick et al [11] extended identity federation into OpenStack identity service with SAML.

In [36], authentication federation trust requirements in different federation strategies such as circle-of-trust, and overlapping federations, has been discussed considering direct and indirect trust relations. The Liberty Alliance Project [9] identified the conceptual framework and guidelines in a circle-of-trust as part of their federated identity vision. Kylau et al [36] trust requirements in authentication federation. Boursas et al [5] present circle-of-trust collaboration trust considera-

tions in authentication federation for assessment of entities' trust outside the circle. The previous research with respect to circle-of-trust has concentrated on authentication federation whereas our scope of contribution lies within authorization federation.

In authorization federation, the issue of granting access to federated users has been discussed in IaaS, PaaS, and SaaS cloud service models. With respect to authorization federation in multi-tenant cloud IaaS, authors [37] provide shibboleth based identity management. Cloud services authenticate users, providing information to service providers, using Shibboleth underlying authentication and authorization with SAML assertions. Moreover, based on privacy policies software delivery model approaches such as Decat et al [16] provide a generic middleware architecture and policy language for federated authorization in SaaS. Thus middleware provides an architecture between requesting tenant and SaaS provider policies to mediate the access with extended XACML policy language [16]. Generally authentication federation has been investigated more extensively than authorization federation which do not deal with different access control models, policies, different service deliveries, and cloud architecture platforms in addition to authenticating federated users. We base our contribution on existing work on authorization federation and scope it to multi-tenant cloud IaaS federation.

2.2 Multi-Tenant Role-Based Access Control

Role-Based Access Control (RBAC) [17] has been dominant access control paradigm in the current computer systems for over two decades. In RBAC, a user access request is granted or denied based on the user role. Furthermore, a role is defined as an abstraction of a set of permissions, and a user acquires only the permissions assigned to its role. In addition to these assignment relations, the user access can be restricted based on authorization policies defined within the enterprise. The first RBAC model, called RBAC96, has been proposed by Sandhu et al [54] as a family of reference models introducing the concepts of role hierarchy and authorization constraints. The ingenuity of RBAC lies within its permission abstraction with roles which makes it policy-neutral and its simplicity to support a wide range of policies. RBAC is simple to perceive and implement policies

while it can support fairly complex and sophisticated policies. It has been rigorously extended beyond its role hierarchy and constraints to support a wide variety of applications in the industry. The majority of cloud platforms today such as OpenStack implements a variation of RBAC to determine access rights. RBAC has been extended towards collaboration across multiple organizations such as ROBAC [64] and GB-RBAC [38]. ROBAC extends RBAC to manage authorizations in multiple organizations by adding both role and organization for authorization decisions. GB-RBAC extends RBAC with group entity to support collaboration in distributed environments. It introduces two level of authorization administration as global and system level, by using system-level and group-level roles.

Current cloud platforms utilize multi-tenancy to segregate underlying shared infrastructure where it has been researched in RBAC since the rise of cloud computing. For instance, authorization model demonstrated in [8] demonstrates a multi-tenant authorization model suitable for cloud supporting multi-tenancy, role-base access control, hierarchical RBAC, path-based object hierarchies and federation.

Further, in cloud IaaS multi-tenancy and collaboration, tang et al [59] propose a cross-tenant trust model (CTTM) in the cloud which encompasses various types of trust relations. Authors identified three types of cross-tenant trust, type α, β, and γ. Collaboration in CTTM is proposed by cross-tenant assignments where trust relations are bridging authorization domains of each tenant. Authors propose a role-based extension (RB-CTTM) where permissions are assigned to roles across tenants based on defined trust types. Trust is defined as type-α where trustor can give access to trustee, type-β where trustee can give access to trustor, and type-γ where trustee can take access from trustor. A motivating example of UTSA and AVIS agreement is given where AVIS has discounted car rental price exclusively for UTSA students. We use this example to explain trust types elaborated in CTTM. In type-α, AVIS by trusting UTSA in type-α can obtain user information in UTSA and assign cross-tenant accesses from UTSA's users to its permissions. In type-α, 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. Type-β trust alters the direction of

the trust relation in α so the trustor (UTSA) can control the exposure of its user information which is necessary for the trustee (AVIS) to make cross-tenant authorization assignments. UTSA by establishing type-β trust with AVIS explicitly exposes its user information to AVIS so that AVIS can assign its permissions to UTSA's users based on UTSA's user information. Type-γ enables both trustor and trustee to control cross-tenant access where by establishing the trust relation, the trustor delegates the control of cross-tenant authorization assignments to the trustee. If AVIS trusts UTSA in type-γ, AVIS delegates UTSA to assign cross-tenant access from UTSA's users to AVIS's permissions.

Moreover, in [58] a multi-tenant role-based access control model is defined enabling authorization in collaborative cloud environments by building trust relations among tenants. The trustee can authorize cross-tenant accesses to the truster's resources consistent with constraints over the trust relation and other components designated by the truster. Trust relation is defined as tenant-trust. If tenant A trusts tenant B then A's issuer exposes A's roles to B's issuer so that B's issuer can assign B's users to A's roles and also B's issuer can assign A's roles as junior roles to B's roles.

2.3 Attribute-Based Access Control

RBAC limitations over the time has been recognized such as role explosion, cost in implementing complex policies, and accommodation of real-time environmental states [14]. Increasingly there is a push to move towards a more general, flexible, and definitive access control model, specifically attribute-based access control model [53].

ABAC determining access to objects by evaluating rules against the attributes of entities, subject and object, on operations, and the environment relevant to a requests [27]. ABAC uses labeled objects and user attributes instead of permissions to provide access control in a flexible manner. If a user has the attributes reflected in the objects requesting access to, then access is granted [14]. This flexibility enables creation of access rules without specifying individual relationships between each subject and each object. Access decisions can change between requests simply by altering attribute values, without requiring changes to the subject or object relationships defining the under-

lying rule sets. Converting access decisions to subject and object attributes with rules for actions in the system provides a more dynamic policy management capability and limits long-term maintenance requirements of object protections. Further, ABAC enables object owners or administrators to apply policy without prior knowledge of the specific subject and for an unlimited number of subjects that might require access [28].

Until recently, there was a lack of consensus on ABAC features and formalized models. Recently, NIST [27] published a guide to ABAC which provides agencies with a definition of ABAC and a description of its functional components. It describes planning, design, implementation, and operational considerations for employing ABAC within an enterprise to improve information sharing while maintaining control of that information. Further, Xin et al [31] proposed a unified ABAC_α model covering DAC, MAC, and RBAC. ABAC_α clearly defines a formalized model with policy configuration points and a policy language. Moreover, authors show that ABAC_α can be used to naturally configure the three classical models. It specifies three components, user, subject, and object associated with corresponding attributes. In ABAC_α, three configuration points are specified for attribute constraint policies and authorization policies where user attributes constrain subject attributes and subject attributes constrain object attributes. Constraints are defined as functions which return true when conditions are satisfied and false otherwise Authorization policies are defined as two-valued boolean functions which are evaluated for each access decision. An authorization policy in ABAC_α for a specific permission takes a subject, an object, and returns true or false based on attribute values. Authors extended to ABAC_β with context attributes covering considerable number of RBAC extensions. We adopt a simplified verism of ABAC_α [31] suitable for our purpose. In particular it eliminates subjects as being distinct from users as is in ABAC_α, and simply treats them to be equivalent.

Another step towards ABAC is adding attributes to roles in RBAC. In order to address a number of RBAC limitations such as initial role structure difficulty, role explosion, inflexibility in rapidly changing domains, Kuhn et al [33] proposed integrating roles with attributes where they identified three approaches, dynamic roles, attribute-centric, and role-centric. Dynamic roles use user and

context attributes to dynamically assign roles to users [29]. In attribute-centric, roles are simply another attribute of users. Since there is no permission-role assignment it discards advantages of RBAC [31]. In role-centric, attributes are added to reduce the permissions available to the user. Jin et al proposed a role-centric attribute-based access control model (RABAC) where authors add attributes to user and object components in RBAC to constrain available permissions. It defines an independent component called the permission filtering policy (PFP), adding to the existing components in RBAC. The PFP, constrains the available set of permissions based on user and object attributes. Motivated by proposed role-centric model in [32], we take a different approach to add attributes to RBAC for multi-tenancy in cloud IaaS collaboration.

Multi-tenancy in ABAC has been researched in different cloud service delivery models. Decat et al [15] introduced Amusa, a middleware for access control management of multi-tenant SaaS applications. Both the provider and the tenants can specify their access rules for the SaaS applications in attribute-based policies. Amusa pre-defines a fixed set of attributes, which the provider can extend for its own application and tenants can build on attributes and policies defined by provider to extend it for their organization (three-layered access control management). Amusa combines the rules of all parties securely and enforces them at run-time. Further, Ngo et al [44] proposed an access control model for multi-tenant cloud services using attribute-based access control model. In this approach access control model is integrated with the cloud infrastructure information description model. Proposed approach generates provider delegation policy automatically from cloud resource descriptions and support multiple levels of delegations for inter-tenant collaborations. Tang [57] specified a multi-tenant attribute based access control enabling cross-tenant access for subjects. Our model differs in structure and cross-tenant access where attribute value assignment provides collaboration.

Also recently, NCCoE [19] published a building block for attribute-based access control, discussing potential security risks facing organizations, benefits from the implementation of an ABAC system and the approach that the NCCoE took in developing a reference architecture and build. Authors discussed ABAC support for business collaboration, following by five steps towards ABAC

and federation architecture.

2.4 OpenStack Cloud Platform

OpenStack is an emerging open-source cloud IaaS platform. Since its introduction in 2010, it has rapidly become popular with enterprises. It has been adopted for a wide range of deployments from public cloud service providers to in-house private clouds. Currently, a wide range of major IT companies participating in its development. We chose OpenStack to implement our models because it is open-source and well accepted in the market.

OpenStack cloud platform consists of interrelated components that control hardware pools of computing, storage, and networking resources where each component is a service in OpenStack communicating with other service with RESTful APIs [18]. OpenStack core services are Keystone (identity service), Nova (compute service), Neutron (networking service), Swift (object storage), Glance (image service), and Cinder (block storage).

Keystone is OpenStack authentication and authorization service. Its core elements are user, group, role, project, domain, and token. A user represents an individual who can authenticate and access cloud resources. In Keystone, groups are set of users in which can be assigned to resources same as users. Roles grant a user or a group set of permitted actions for either a specific project or an entire domain. Each role is paired with a project or a domain in time of assignment to users or groups. To that end, roles are associated with users in relation to projects as project-role-pairs (PRP). A project is the base unit of ownership in OpenStack and each resource is owned by a specific project while each project is owned by a domain. A project represents a tenant in OpenStack. A domain is an administration scope of users, groups, and projects. Each user, group, and project is owned by exactly one domain. In Keystone, token represents the authenticated identity of a user granting authorization on a specific project or domain. In [60], OpenStack access control model is identified with respect to its core elements and operations.

Keystone is organized as a group of internal services as identity, resource, assignment, token, catalog, and federation. We focus on its current state towards federation. The identity service

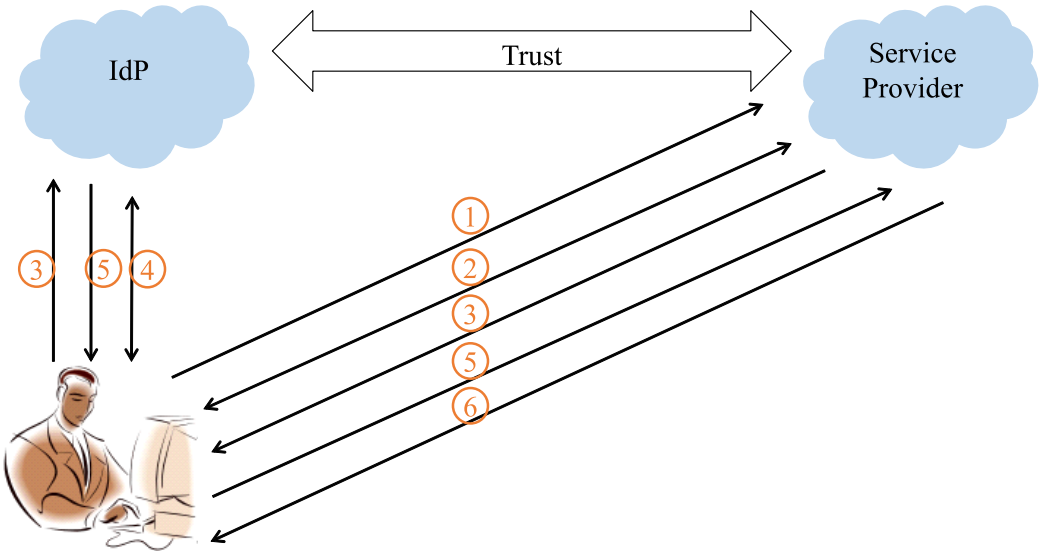


Figure 2.1: Adding Federated Identity Management to OpenStack.

provides authentication validation and management of users and groups data. Resource service manages data about projects and domains. Assignment service provides data about roles and role assignments to the entities managed by the identity and resource services. Token service validates and manages, tokens that are used for authenticating requests once a user's credentials verified. Two types of tokens are implemented in Keystone (In Juno release) PKI and UUID. In this dissertation's implementations, we use PKI token format where each PKI token maintains user credentials, target projects, assigned roles to projects and service catalogs in its payload. Catalog service provides endpoint registry and endpoint discovery mechanisms in Keystone. Policy service provides rule-based authorization engine used for implementing policies in Keystone. Federation service provides federation APIs including registering identity providers, service providers, protocols, and mappings in addition to create and consume SAML assertions.

Keystone federation service enables Keystone to provide SAML assertions and consume these assertions. In Icehouse release [49], Keystone federation service added to federate OpenStack with an identity provider (Idp), meaning a trusted Idp was able to federated its users to access resources in OpenStack. In [11], Identity federation to OpenStack is discussed. Figure 2.1 depicts such enforcement model which is based on OpenStack Icehouse federation API. In this model, an Idp is federated to OpenStack Service Provider enabling Idp's users to access resources in OpenStack

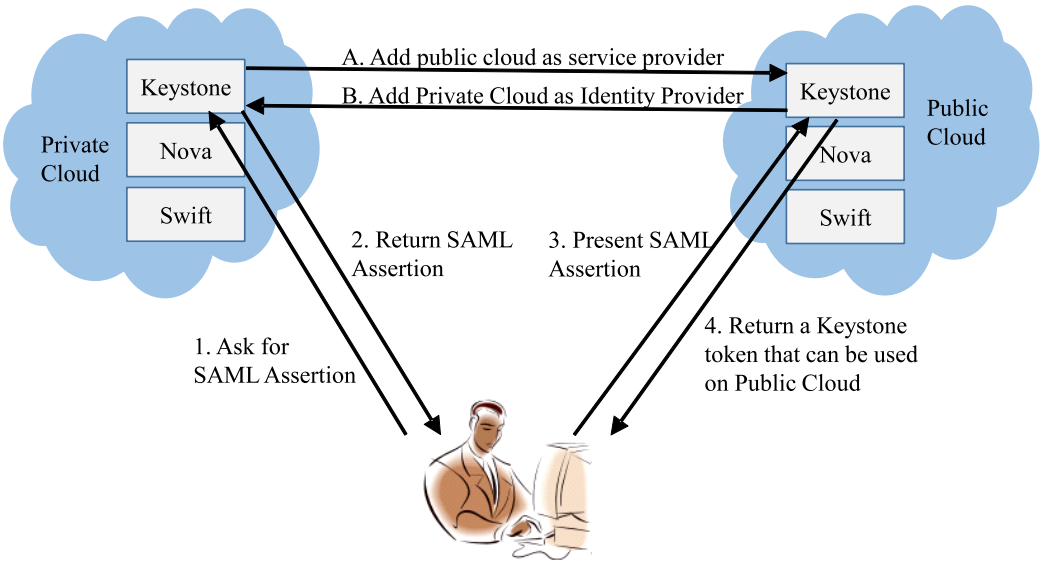


Figure 2.2: Keystone to Keystone Federation.

while they are authenticating to their local Idp. Figure 2.1 enforcement is as following: (1) User requests to access a service in Service Provider. (2) Service Provider determines user Idp. (3) User is redirected for authentication to it's Idp. (4) User presents its credentials and authenticate to its local Idp. (5) Idp redirects users' attributes. (6) SAML assertions are presented to Service Provider, from this stage user is mapped to local user or group and act as normal user in OpenStack.

Since OpenStack Juno release, OpenStack is capable of generating SAML assertion enabling it provide Keystone to Keystone federation. Figure 2.2 depicts Juno Keystone to Keystone federation in OpenStack.

Current Keystone federation service includes following APIs, Identity Provider, Protocols, Mappings, and Service Provider. Identity Provider (Idp) and Service Provider (SP) holds information about trusted Idps to accept assertions from and trusted SPs to send assertions. These information includes protocols, IP addresses, and descriptions about Idp and SP. Protocol contains information that dictates which mapping rules to use for a given incoming request. An IdP may have multiple supported protocols. Mapping is a set of rules to map federation protocol attributes to Identity API objects. It is used to map federated credentials to local credentials.

Chapter 3: FEDERATION FRAMEWORK FOR CLOUD

This chapter introduces our cloud federation framework elaborating our perspective of federation and multi-cloud in the context of our scope of homogeneous multi-tenant multi-cloud IaaS. We define our federation framework with Peer-to-Peer and Circle-of-Trust federation models followed by tenant-trust and defined types of trust respectively. Finally, our scope of study and a taxonomy of contributions in this dissertation is discussed in the context of this framework.

3.1 Federation

In real life, collaboration among organizations is inevitable due to growing challenges of global competition, rapid changes and increasing complexity of organizational structures. Organizations should be able to quickly come together and collaborate to solve a specific problem or exploit a specific opportunity. Such a group of collaborative organizations establishes a federation. A federation can be defined as an organizational structure where multiple organizations have set up collaborative agreements. Each organization has a separate administration and domain connected to other organizations by trust agreements. The concept of federation has a long and varied history in computer systems. Just as one example, the notion of virtual organizations has been developed in the distributed systems and grid computing communities going back almost two decades. It is beyond the scope of this dissertation to give a comprehensive review of federation in computer systems, rather our focus is on cloud federation. For simplicity we will henceforth understand the term federation to mean cloud federation. Cloud federation is a multi-faceted concept and has been treated in different ways in the literature. A cloud federation can be defined as a collaboration of cloud service providers and identity providers in order to share their services and resources within participating clouds based on trust agreements.

In the following we characterize cloud federation compliant with NIST definition of cloud computing [40]. We can distinguish cloud federations with *service*, *platform*, *trust*, and *identity* properties in a federation, illustrated in Figure 3.1.

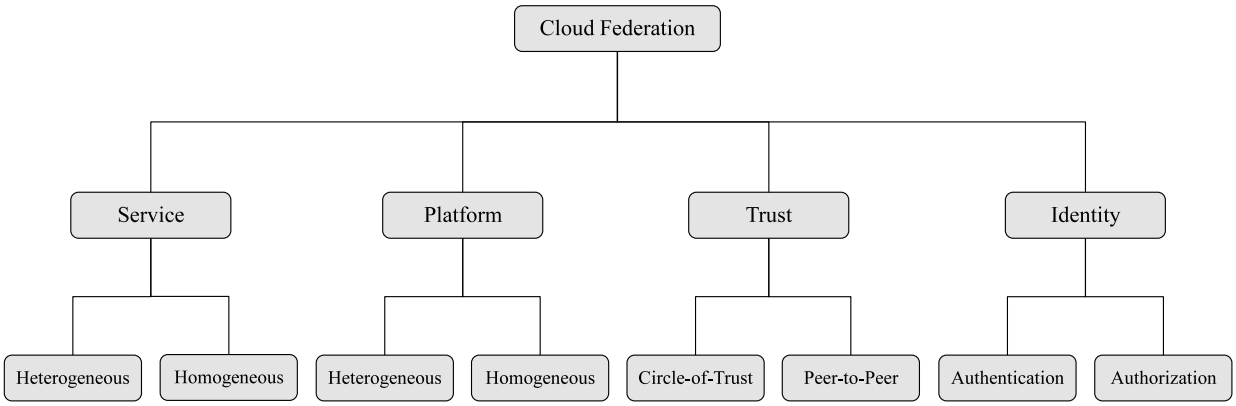


Figure 3.1: Characteristics of Federation in Cloud Computing.

Cloud federation offers collaboration between different service layers,i.e., Software, Platform, and Infrastructure-as-a-Service in the cloud [40]. In IaaS, offered services are processing, storage, networks, and other fundamental computing services which are homogeneous computing services. However, SaaS and PaaS span a wide range of services from provider's applications running on a cloud infrastructure to consumer-created applications using programming languages supported by provider which are typically heterogeneous services [40].

Cloud computing offers different deployment models such as private, public, and hybrid [25] where cloud platforms support different policy models. Cloud federation between homogeneous deployments such as two OpenStack public or two OpenStack private clouds forms a homogeneous federation. On the contrary, federation of an OpenStack private cloud with a proprietary public cloud such as AWS or Azure would form a heterogeneous cloud federation with respect to platform.

Trust in cloud federation defines contracts specifying obligations and rights each party has and policies to follow. Two fundamental federation models arise in cloud federation trust models first, where tenants establish trust directly between each other and second, where trust is established by a group of tenants. In Peer-to-Peer trust agreement trust is established between two tenants directly, as depicted in Figure 3.2. In the commercial setting, federating tenants with Peer-to-Peer trust is more common due to limited trust, and presumably enhanced security. Moreover, establishing Peer-to-Peer merely requires collaborating tenants' trust negotiation. Further, in a

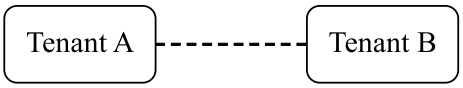


Figure 3.2: Peer-to-Peer Federation Model.

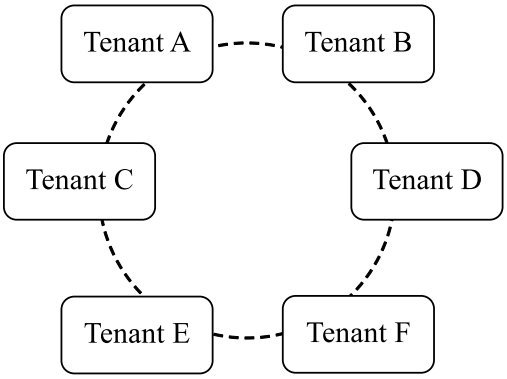


Figure 3.3: Circle-of-Trust Federation Model.

group of tenants where trust is established across an association of tenants called Circle-of-Trust, illustrated in Figure 3.3. In a Circle-if-Trust, tenants adhere to policies and contracts established in the circle to collaborate. To become a circle member, tenants are required to adhere to the tenant-trust specifications, in particular to demonstrate that circle policies are respected and enforced. Moreover, based on circle governance tenants requesting to join the circle must be accepted by circle tenants. In this dissertation, we consider collaborative governance, where all the tenants share the governance of the circle (see section 6.1). For instance, CERN is a Circle-of-Trust cloud federation in which institutions joining the circle of institutions can gain certain access to analytic data and computation resources across their home tenants.

In cloud federation scenarios, users are federated to access services and resources outside their home tenant adminstration domain. Identity federation aims at authenticating and authorizing users across their resident tenants. Authentication federation process allows a single user to authenticate across multiple IT systems or even organizations, such as single sign-on. Authorization federation aims at determining federated users' permissions to access federated resources and services. In this dissertation, our focus is on authorization federation in multi-tenant cloud IaaS. In authorization federation, both trustor and trustee tenants must be allowed to control federated users and resources in a trust relation. For instance, if tenant A trusts tenant B, tenant A is trustor tenant and tenant B

is trustee tenant. We further discuss these concepts in the following sections.

3.2 Multi-Cloud

In this section, we elaborate our perspective on cloud federation and multi-cloud. In general, cloud federation can be of interest to customers as well as cloud service providers. Cloud consumers profit from lower cost, performance improvement, and more sophisticated services while service providers benefit from collaboration scenarios such as cloud bursting and load balancing. Common terms used for federation in cloud are multi-cloud, cloud federation, hybrid cloud, and inter-cloud. First, we describe our view of cloud federation and multi-cloud, specifically scoped to our focus in authorization federation IaaS, then we briefly present a taxonomy of federation in cloud IaaS.

In this dissertation, we adopt on the following meanings for cloud federation and multi-cloud. *Cloud Federation* is a federation of cloud service providers and identity providers to share services and resources based upon trust agreements. *Multi-Cloud* is a federation of multiple cloud service providers (public, private, or hybrid) within different administrative domains (Cloud and Domain) to provide complex services at specified service model (Infrastructure, Platform and Software). In the rest of this dissertation, we adhere to these definitions.

The difference between cloud federation and multi-cloud is denoted by the degree of collaboration and by the way users interact with the cloud. In our view, cloud federation is more seamless collaboration compare to multi-cloud. Cloud federation services are shared without user consensus, meaning user interacts with one cloud and is unaware of resources and services origin such as cloud bursting or provisioning VMs' location whether it is in the home tenant or a federated tenant. In contrast, in multi-cloud federation users are aware of being federated to another cloud, using client or cloud APIs for federation. In multi-cloud federation user can choose to be federated with proper credentials. Moreover, in cloud federation, collaboration type is voluntary and there is an agreement between clouds to share resources whereas multi-cloud does not imply voluntary interconnection and sharing of providers' infrastructures.

Hybrid cloud is defined by NIST as "a composition of two or more distinct cloud infrastruc-

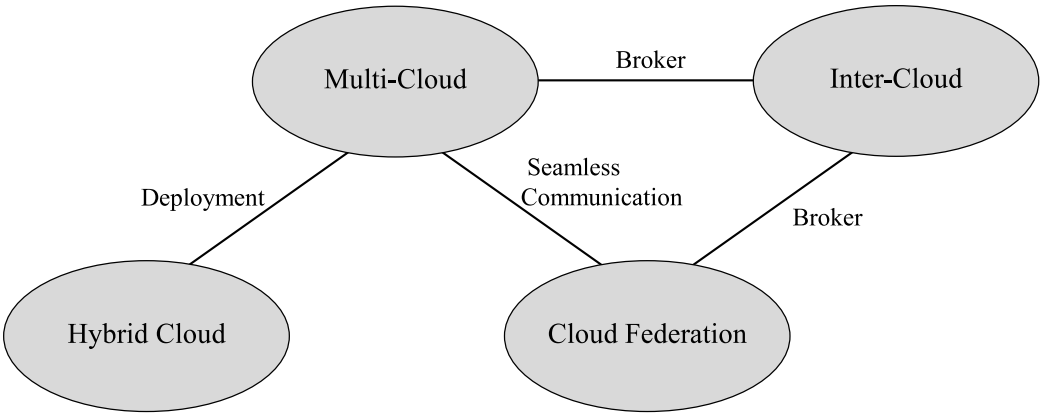


Figure 3.4: Classification of Federation in Cloud .

tures (private, community, or public)" [40]. We consider hybrid cloud as type of multi-cloud that connects clouds in terms of their deployment models. Multi-cloud is a more general term in comparison to cloud federation and hybrid cloud.

Inter-cloud definitions are commonly coupled with a broker agent or a broker service for reassignment of services and workload for the purpose of performance considerations. Generally, a broker refers to a service that acts as a client to provision resources and deploy application components. We identify an inter-cloud as a cloud federation or multi-cloud model with a broker service that offers dynamic service provisioning. Figure 3.4 illustrates our conceptual classification of federation in cloud IaaS. Within our scope, we consider federation in the multi-tenant homogeneous cloud platforms where each cloud is distinct and remains autonomous. In this framework, we focus on authorization federation considerations across tenants.

Administrative Domains in Multi-Cloud

In authorization federation, trust defines administrative authority granted to each party over trust relation and federated assignments. In this concept, the domain in which trust is established identifies resources federating with the specified trust relationship such as services, resource containers, operations, or data objects. Thus, we identify two administrative domains in cloud federation, cloud and tenant. To clarify this concept, we illustrate conceptual trust in Figure 3.5 in an OpenStack cloud platform. In OpenStack domain represents tenant concept.

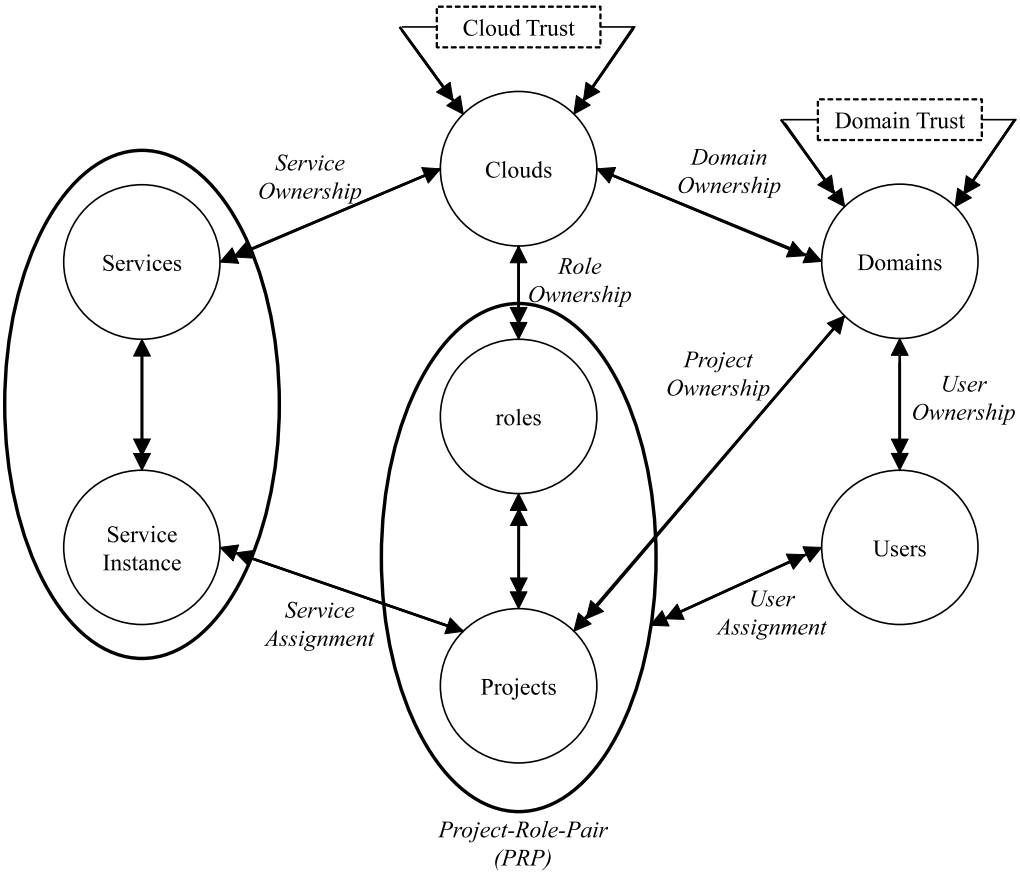


Figure 3.5: Administrative Domains in an OpenStack Cloud

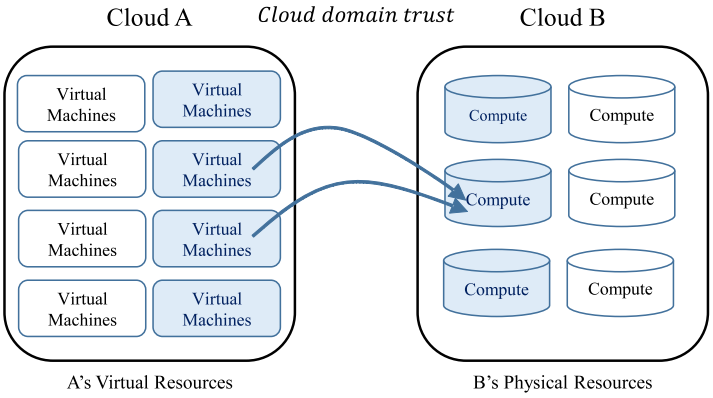


Figure 3.6: Resource Allocation in Cloud-Trust.

In an IaaS cloud system, *cloud* domain holds authority over services such as compute, storage, network, and identity, as well as over lower administrative domains such as tenants. In a cloud-trust between two clouds, services are federated aiming at load balancing within a multi-cloud federation. Cloud bursting scenarios are of cloud service providers interest, for example a private cloud can establish a cloud-trust between its private cloud and a public cloud provider to perform resource allocation and outsourcing in peak usage or when it is running out of resources. Figure 3.6 depicts such service federation in cloud-trust.

A *tenant* is an administrative domain of resources in a cloud such as users, groups and projects in OpenStack. In a tenant-trust users and resources are federated through assignments. In tenant domain, with proper tenant-trust, users are enabled to access resources beyond their home tenant. In this dissertation, we define a set of tenant-trust relations in tenant domain to enable user-role and attribute assignments in Peer-to-Peer and Circle-of-Trust federation models. In the following section we discuss tenant-trust relationships in cloud IaaS.

3.3 Cloud Federation Framework

The building block of federation is trusted relationship. We elaborate trust between tenants in multi-tenant multi-cloud platforms. Federation patterns defined are solely based on direct trust where trust relationship exists between two tenants. In Peer-to-Peer federation, a tenant trusts

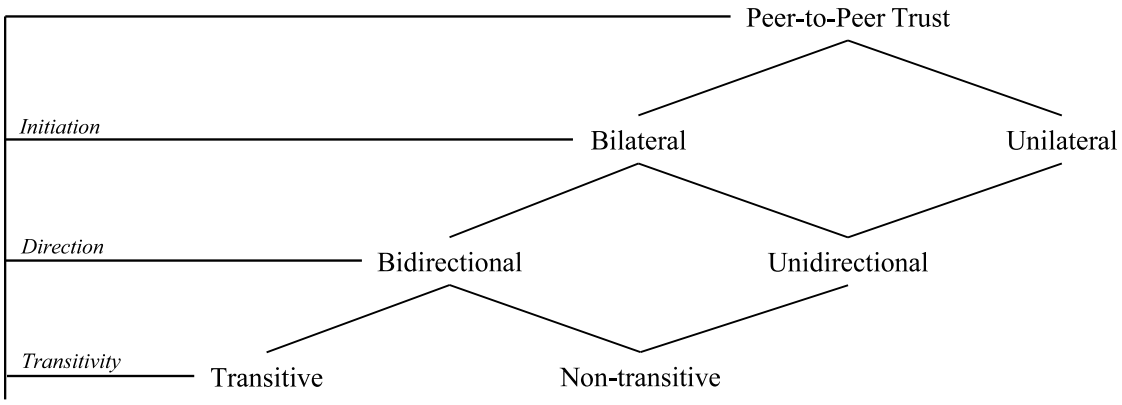


Figure 3.7: Peer-to-Peer Trust Characteristics.

(trustor tenant) another tenant (trustee tenant) directly, similarly in Circle-of-Trust a tenant trusts a group of tenants directly. In our scope a direct trust relationship exists between tenants in Peer-to-Peer trust and Circle-of-Trust. In a Circle-of-Trust every two tenants only belong to one circle without overlapping circles where two tenants trust each other through more than one circles. Although indirect trust and overlapping circles can be easily derived from federation patterns in this dissertation.

Peer-to-Peer federation pattern is most common collaboration model in industry, for instance, current cloud platforms such as OpenStack and AWS implements Peer-to-Peer federation in their cloud APIs. Peer-to-Peer (P2P) model can be extended to cover more complicated types of federation such as undirected delegated trust (where a trust relationship does not exist directly between tenants). In a Peer-to-Peer federation, trust relationship is characterized by *initiation*, *direction*, *transitivity* properties between two tenants as depicted in Figure 3.7.

P2P-Initiation (Bilateral vs. Unilateral). Trust initiation is regarding agreement on trust establishment. In a bilateral trust, both tenants are required to confirm trust relation whereas in unilateral trust, trustee tenant's consent is not required. In tenant administration domain, unilateral trust is preferred for simplicity and ease of administration and operation while in cloud administration domain, trust is required to be bilateral, e.g., for cloud bursting scenarios both clouds consent is required.

P2P-Direction (Bidirectional vs. Unidirectional). In a bidirectional trust relation both par-

ticipating tenants are equally enabled by the trust. Conversely, in a unidirectional trust, the actions are available only on one side or the other. We note that bidirectional trust in fact comprises two unidirectional trust relations, so thereby we can extend unidirectional trust to bidirectional easily. In Peer-to-Peer federation unilateral trust typically leads to a unidirectional trust because trustee consent is not required enabling both tenants equally is not reasonable, on the other hand bilateral Peer-to-Peer trust can be essentially bidirectional or unidirectional.

P2P-Transitivity (Transitive vs. Non-Transitive). In a Peer-to-Peer federation we consider direct or non-transitive trust. Transitive or indirect trust is characterized by the fact that there is not a direct trust relationship between two tenants. Instead both tenants have a trust relationship with a common third tenant. This relationship leverages trustworthiness of unknown federated partners based on mutual trust with a third trusted partner. Unidirectional trust is essentially non-transitive due to the fact that directed trust is unilateral and trustee, and trustor tenants are not equally authorized in Peer-to-Peer federation.

In this dissertation, tenant-trust in Peer-to-Peer federation model is considered as unilateral, unidirectional and non-transitive trust relationship. Also, tenant-trust is reflexive meaning each tenant trusts itself.

In a Circle-of-Trust, trust relationships are defined between all circle tenants. We use terms entities and principals interchangeably for tenants. Tenants make assertions (user-role assignments) in the circle, assigning users to roles. Circle-of-Trust federation eliminates the need to create multiple trust relationships to collaborate, in addition to simpler administration in contrast to multiple Peer-to-Peer trust relationships. Circle-of-Trust is applicable to a set of tenants which follow similar federation policies and are to some extent authorized to make assertions in the circle similarly. Further, we discuss in heterogeneous circles that assertions can be limited based on tenant type. Tenants in the circle adhere to similar authorization and rules to collaborate.

We distinguish trust relationships in the Circle-of-Trust (CoT) with the following properties, *entity coupling*, *initiation*, *direction*, and *transitivity*. Figure 3.8 gives a logical hierarchy of these trust properties discussed below. Vertical placement of characteristics is selected to better illustrate

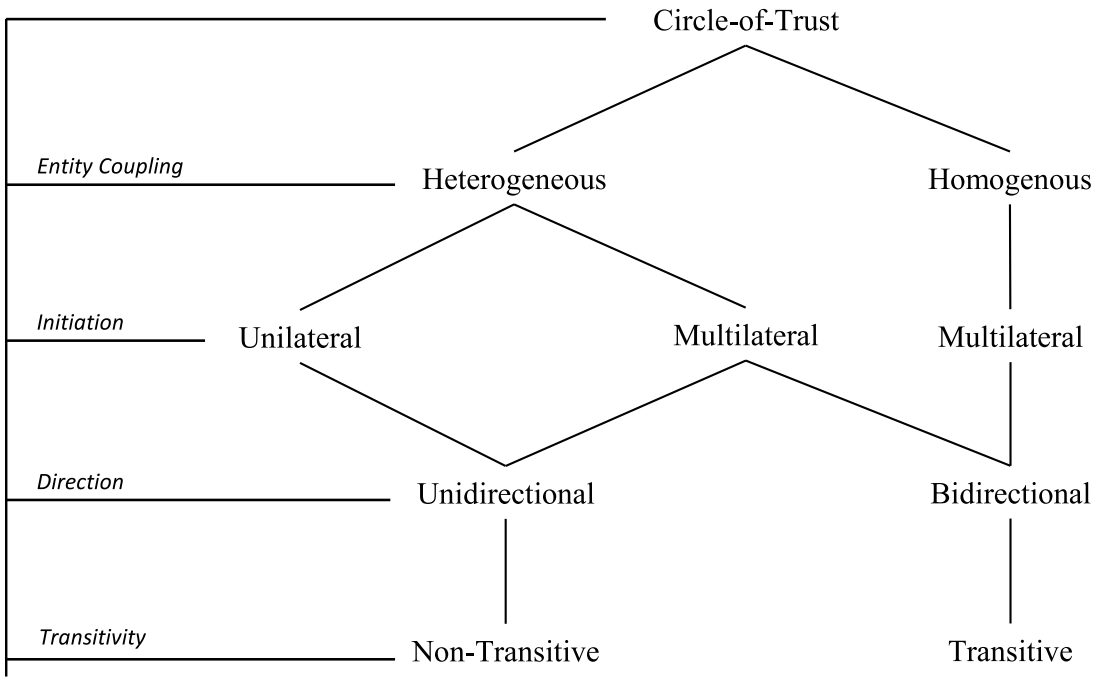


Figure 3.8: Circle-of-Trust Characterization.

trust relations in our scope of contribution.

CoT-Entity Coupling (Homogeneous vs. Heterogeneous). In a circle-of trust, type of entities engaging in interactions determines homogeneity or heterogeneity of the circle, shaping its authorized interactions between tenants. Moreover, with each circle type a set of trust properties are applicable. By *homogeneous circle* we denote the case where entities are uniform. For instance a circle of universities forms a homogeneous circle. In a homogeneous circle, collaborating principals are equally authorized to make cross-tenant authorization assertions. A *heterogeneous circle*, is an association of non-uniform entities where each type of entity is authorized specifically to make certain assertions. For instance, a circle consisting of universities, insurance companies, and banks establishes a heterogeneous circle. In this scenario, universities can assign users to discounts in insurance companies while insurance companies cannot assign users to resources in the universities. In this dissertation we use type and domain interchangeably denoting the type of entities in a heterogeneous circle.

CoT-Initiation (Multilateral vs. Unilateral). If trust initiation to join a circle is required to

be confirmed by all circle members, trust is considered *multilateral*. In special situations when joining members are not authorized to make assertions (in heterogeneous circles) trust initiation is not required to be confirmed by all circle members denoted as *unilateral* trust. For instance a domain of insurance companies joins a heterogeneous, unilateral circle of institutions. Insurance entities in the circle are not authorized to make assertions whilst institution entities are authorized to assert their users to discounts available to universities.

CoT-Direction (Bidirectional vs. Unidirectional). In a circle direction of trust determines whether both participating circle members have equal authorizations or only one side is authorized to make assertions. If partners are authorized equally to make assertions, trust relation is *bidirectional*, otherwise it is *unidirectional* trust. Homogeneous circles' relations are bidirectional while heterogeneous circles support both trust directions. Unilateral heterogeneous circles such as given example above are only unidirectional in trust relations. Circle of universities is an example of bidirectional trust in a homogeneous circle. Sharing files in Dropbox is an example of a unidirectional trust where a user can share files with a group of users unidirectionally.

CoT-Transitivity (Transitive vs. Non-transitive). In a homogeneous circle, bidirectional trusts are essentially transitive where all members trust and are trusted by other circle members. In heterogeneous unidirectional circles, trust relations cannot be transitive. For example in heterogeneous unidirectional circle of institutions, considering banks and insurance companies, an institution can assign students to bank specific account types in banks whilst banks can assign employees to health insurances in insurance companies. Considering heterogeneous domains in the circle, a university trusting a bank and a bank trusting an insurance entity does not necessarily imply that the university can assign students to insurance.

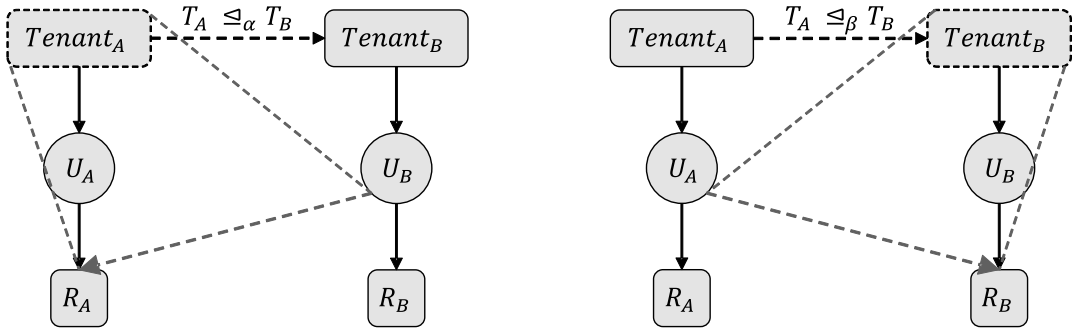
In this dissertation, we consider multilateral, bidirectional, and transitive trust relationship for homogeneous circles. Trust relations between tenants in heterogeneous circle are considered multilateral, unidirectional, and non-transitive. In the following we identify how trust relations authorize cross-tenant assignments in Peer-to-Peer and Circle-of-Trust federation models.

3.4 Tenant-Trust Framework

In this section, we review tenant-trust. We define tenant-trust in Peer-to-Peer and Circle-of-Trust federation models. In Peer-to-Peer trust we enable user-role assignments and attribute assignments across tenants with four trust types respectively. In Circle-of-Trust we define two trust types enabling user-role assignments in homogeneous and heterogeneous circles.

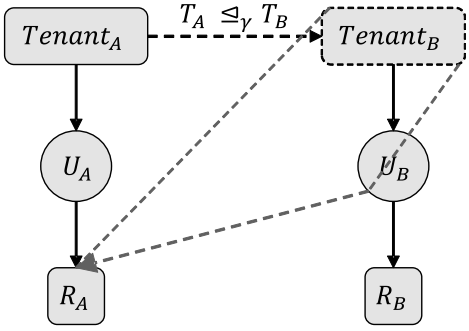
We distinguish tenant-trust in Peer-to-Peer with role-based and attribute-based federation. In Peer-to-Peer role-based federation model, various trust types have been elaborated. Recently in [59] set of three cross-tenant trust models type α, β, and γ has been defined. We extend these definitions to multi-cloud environment in addition to introducing a new trust type called δ. We use "$\trianglelefteq$" to show tenant-trust between two tenants in Peer-to-Peer federation where $T_A \trianglelefteq T_B$ signifies that tenant A trusts tenant B. In a Peer-to-Peer federation, trustor can take permission to make user-role assignments or it can give permission to trustee to make user-role assignments. If trustor gives permission to trustee to make user-role assignments, three scenarios arises. First, trustor authorizes trustee to assign its users to roles in trustee. Second trustor authorizes trustee to assign trustee's users to its roles. Third trustor grants permission to trustee to make user-role assignments in its tenant. Our tenant-trust enables user-role assignment in multi-tenant multi-cloud IaaS. To that end, a prior trust in cloud administrative domain is required. We elaborate tenant-trust in multi-cloud environment with a cloud-trust existing between corresponding tenants prior to establishing tenant-trust. Moreover, trust relations are defined as unilateral, unidirectional, and non-transitive, and in our tenant-trust, trustor tenant can establish and end trust relationship.

Type-α, perhaps the most intuitive form, elaborates trustor to take permission to share resources with a trusted tenant where it assigns trustee users to its roles, illustrated in Figure 3.9a. Currently in OpenStack cloud platform this type of trust exist between tenants where a domain (OpenStack tenant) can assign users from other domains to its project-role-pairs (in OpenStack projects are associated with roles as project-role-pairs). In multi-cloud situation, type-α tenant-trust can be utilized as a trust relationship between identity provider and service provider. If service provider

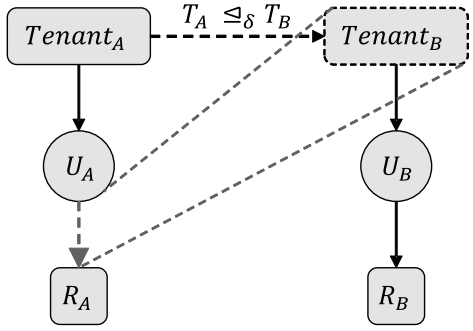


(a) Tenant-Trust Type-α.

(b) Tenant-Trust Type-β.



(c) Tenant-Trust Type-γ.



(d) Tenant-Trust Type-δ.

Figure 3.9: User-Role Assignment in Peer-to-Peer Tenant-Trust.

trusts identity provider, then identify provider can federate its users to access service provider's resources. Type-α tenant-trust is defined in Table 3.1.

In type-β, trustor gives permission to trustee to assign trustor's users to trustee's roles. It is shown in Figure 3.9b. Trustor tenant trusts trustee tenant by exposing trustee's user set for cross-tenant user assignments. Type-β allows resource owner to determine access to its resources. For instance, a bank trusts another bank, in this type of trust trustees are not obliged to expose any resources to trusters as well as delegating other tenants permission to make user-role assignments. It is defined in Table 3.1.

Type-γ grants permission to trustee tenant to assign its users to roles in trustor tenant. It is illustrated in Figure 3.9c. In type-γ, trustor's role structure is not sensitive for trusted tenants' user assignments, meaning if a tenant trust another tenant, it trust the trustee tenant with exposing its resources. For example, an insurance company trusting an institute with type-γ grants institute's tenant to assign students and employees to promotions in its company. Type-γ is defined in

Table 3.1: User-Role Assignment in Peer-to-Peer Tenant-Trust Types.

Tenant-Trust	Definition
Tenant-Trust Type-α:	*If $tenant_A \trianglelefteq_\alpha tenant_B$, then tenant A is authorized to assign tenant B's users to it's roles. Tenant A controls user-role assignments.*
Tenant-Trust Type-β:	*If $tenant_A \trianglelefteq_\beta tenant_B$, then tenant B is authorized to assign tenant A's users to its roles. Tenant B controls user-role assignments.*
Tenant-Trust Type-γ:	*If $tenant_A \trianglelefteq_\gamma tenant_B$, then tenant B is authorized to assign its users to tenant A's roles. Tenant B controls user-role assignments.*
Tenant-Trust Type-δ:	*If $tenant_A \trianglelefteq_\delta tenant_B$, then tenant B is authorized to assign users to roles in tenant A. Tenant B controls user-role assignments.*

Table 3.1.

Another type of trust is type-δ where trustor grants permission to make user-role assignments in its tenant. Type-δ trust is defined in Table 3.1 and depicted in Figure 3.9d. This type if trust delegates administrative privileges to trussed tenants. For instance, two distinct tenants' administration can be merged virtually using type-δ trust, specifically in interest of organizations in time of reorganizing the overall structure of the organization.

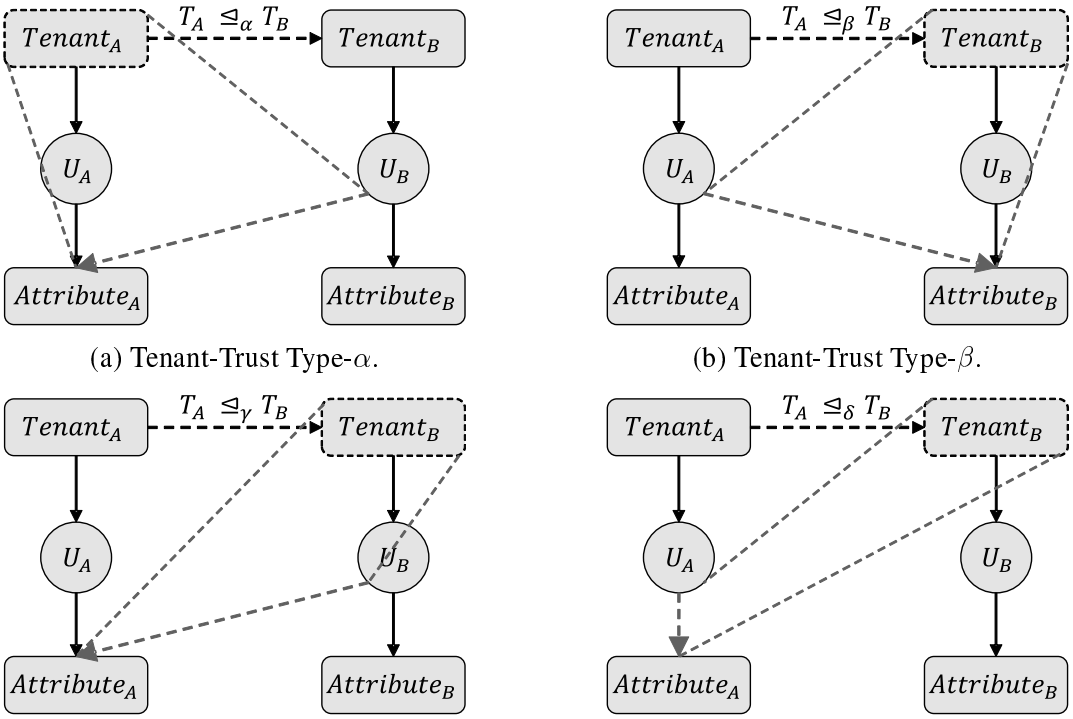
In Peer-to-Peer federation, we also define tenant-trust with attribute assignment where a tenant trust another tenant to either grant or take permissions to assign attribute values to users. Tenant-trust types with attribute assignment are defined in Table 3.2. Trust relationships defined in Peer-to-Peer attribute-based model are unilateral, unidirectional, and none-transitive trust relationships. We use "$\trianglelefteq$" to show trust between two tenants. In Peer-to-Peer attribute-based model trust relationship is required to be reflexive but is not required to be symmetric or antisymmetric. Multi-tenant attribute-based access control model (MT-ABAC) employs Peer-to-Peer tenant-trust which we explain in detail in Chapter 5 to enable tenant domain cloud federation. In MT-ABAC each user is associated with a set of attributes called user attributes, consumed in evaluating users to grant access to resources in the tenant. By enabling attribute assignment across tenants resources are shared in cloud IaaS.

Table 3.2: User-Attribute Assignment in Peer-to-Peer Tenant-Trust Types.

Tenant-Trust	Definition
Tenant-Trust Type-α:	*If $T_A \trianglelefteq_\alpha T_B$, Tenant T_A is authorized to assign values for T_A's user attributes to Tenant T_B's users. Tenant T_A controls cross-tenant attribute assignments.*
Tenant-Trust Type-β:	*If $T_A \trianglelefteq_\beta T_B$, Tenant T_B is authorized to assign values for T_B's user attributes to Tenant T_A's users.T_B controls cross-tenant attribute assignments.*
Tenant-Trust Type-γ:	*If $T_A \trianglelefteq_\gamma T_B$, Tenant T_B is authorized to assign values for T_A's user attributes to Tenant T_B's users. Tenant T_B controls cross-tenant attribute assignments.*
Tenant-Trust Type-δ:	*If $T_A \trianglelefteq_\delta T_B$, Tenant T_B is authorized to assign values for T_A's user attributes to Tenant T_A's users. Tenant T_B attribute assignments in tenant T_A.*

We define a set of four attribute assignments tenant-trust relationships, α, β, γ, and δ. In type-α, trustor tenant is authorized to assign attribute values to users in trustee tenant depicted in Figure 3.10a. Type-β tenant-trust grants permission to trustees to assign attribute values to trustee tenants illustrated in Figure 3.10b. In type-γ, trustee is granted permissions to make attribute assignment to its users from trustor tenant attribute values shown in Figure 3.10c. Finally type-δ grants attribute assignment ins trustor tenant to trustee. Type-δ conceptual attribute assignment is demonstrated in Figure 3.10d.

In a Circle-of-Trust, tenants establish trust with all tenants in the circle by joining the circle. We distinguish two types of circles, homogeneous and heterogeneous based on uniform and non-uniform tenant types respectively. In homogeneous circles, tenant-trust is multilateral, bidirectional, and transitive relationship and in heterogeneous circles, tenants trust each other with a multilateral, bidirectional, and non-transitive relationship. Moreover, trustor and trustee is no longer valid in Circle-of-Trust federation model since all tenants trust each other, trustor and trustee terms are not meaningful. we distinguish tenants by user-owner tenant and resource-owner tenants.



(a) Tenant-Trust Type-α.

(b) Tenant-Trust Type-β.

(c) Tenant-Trust Type-γ.

(d) Tenant-Trust Type-δ.

Figure 3.10: Attribute Assignment in Peer-to-Peer Tenant-Trust.

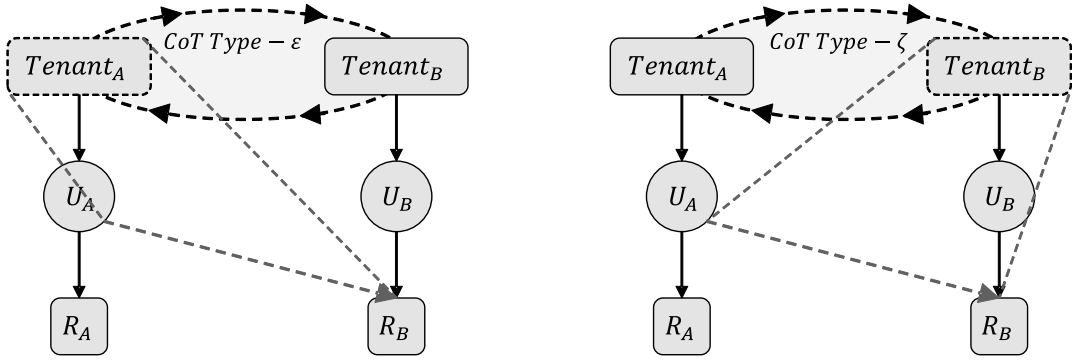
Also, we define tenant-trust across a circle to mean the same tenant-trust applies to all tenants' relationships in the circle. User-role assignments in Circle-of-Trust is defined as user to public role assignments which is described in Chapter 6 in detail.

We define two types of trust in Table 3.3, type-ϵ where user-owner tenants can assign their users to public roles in the trusted tenants illustrated in Figure 3.11a and type-ζ where resource-owner tenants can assign users in the circle to their public roles demonstrated in Figure 3.11b. We use "$\lhd$" to represent tenant-trust relationship between two tenants which are members of the same circle.

Type-ϵ enable tenants in the circle to assign their users to roles of other tenants in the circle. The advantage of this type of tenant-trust is its simplicity to administer and implement as long as tenants' resources shared are not sensitive within the circle and tenants are willing to delegate user-role assignments to trusted tenants in the circle. For instance, an academic Circle-of-Trust is a motivation of this type of circles where academic tenants establish a Circle-of-Trust to share

Table 3.3: User-Role Assignment in Circle-of-Trust Tenant-Trust Types.

Tenant-Trust	Definition
Tenant-Trust Type-ϵ:	*If $T_A \lhd_\epsilon T_B$, Tenant T_A is authorized to assign its users to T_B's roles. Tenant T_A controls user assignments.*
Tenant-Trust Type-ζ:	*If $T_A \lhd_\zeta T_B$, Tenant T_B is authorized to assign T_A's users its roles. Tenant T_B controls user assignments.*



(a) Tenant-Trust Type-ϵ. (b) Tenant-Trust Type-ζ.

Figure 3.11: User-Role Assignment in Circle-of-Trust Tenant-Trust.

computing resources. Any academic tenant can assign its users to resources across tenants in the circle.

Type-ζ on the other hand, follows a different purpose to protect shared resources where user-role assignments are administered by resource-owner tenants. Tenants do not want to delegate trusted tenants permission to make assertions to their shared resource. A Circle-of-Trust of financial companies is a motivating example of type-ζ tenant-trust. Financial companies do not want to expose their resources to collaborating tenants in the circle since their resource are highly sensitive even within trusted tenants in the circle. We further explain Circle-of-Trust tenant-trust in homogeneous and heterogeneous circles in Chapter 6.

3.5 Scope of this Dissertation

Our contributions in this dissertation scopes to authorization federation in homogeneous multi-tenant multi-cloud IaaS platforms with Peer-to-Peer and Circle-of-Trust federation models. Figure 3.12 illustrates our scope of contribution. We distinguish federation in cloud with service, platform, trust, and identity properties.

Service (Iaas vs PaaS vs SaaS). Cloud platforms offer services at IaaS, PaaS, and SaaS, where services in IaaS layer are homogeneous. In SaaS and PaaS federation spans a range of heterogeneous services. We focus on IaaS cloud service layer where services are homogeneous.

Platform (Heterogeneous vs Homogeneous). Deployment models in federation include homogeneous and heterogeneous cloud platforms.We consider homogeneous cloud platforms in cloud IaaS federation. To that end, models discussed in this dissertation in multi-tenant cloud IaaS can be easily extended to multi-tenant multi-cloud IaaS as we scope to homogeneous cloud platforms.

Trust (Circle-of-Trust vs Peer-to-Peer). In a cloud federation two fundamental federation trust models are identified as Circle-of-Trust and Peer-to-Peer where we elaborated these federation models with tenant-trust. In this dissertation, trust is defined between tenants as tenant-trust in the context of Peer-to-Peer and circle-of-Trust federation models. In our scope, we identify that Peer-to-Peer model trust is established between tenant pairs while in Circle-of-Trust relationships are established between a group of tenants. In a Peer-to-Peer trust we scope to role-based and attribute-based access control models and in Circle-of-Trust we consider role-based and role-centric attribute-based models with cross-tenant assignments.

Identity (Authentication vs Authorization). Identity federation deals with authenticating and authorizing federated user beyond their home tenant. In IaaS cloud context authentication process identifies users identity and authorization considers set of permissions granted with users' credentials. In this dissertation, we focus on authorization federation in cloud IaaS.

In our scope, we consider multi-cloud federation as distinct tenants collaborating in multi-cloud and cloud platforms. Moreover, in tenant administrative domains (cloud and tenant domain

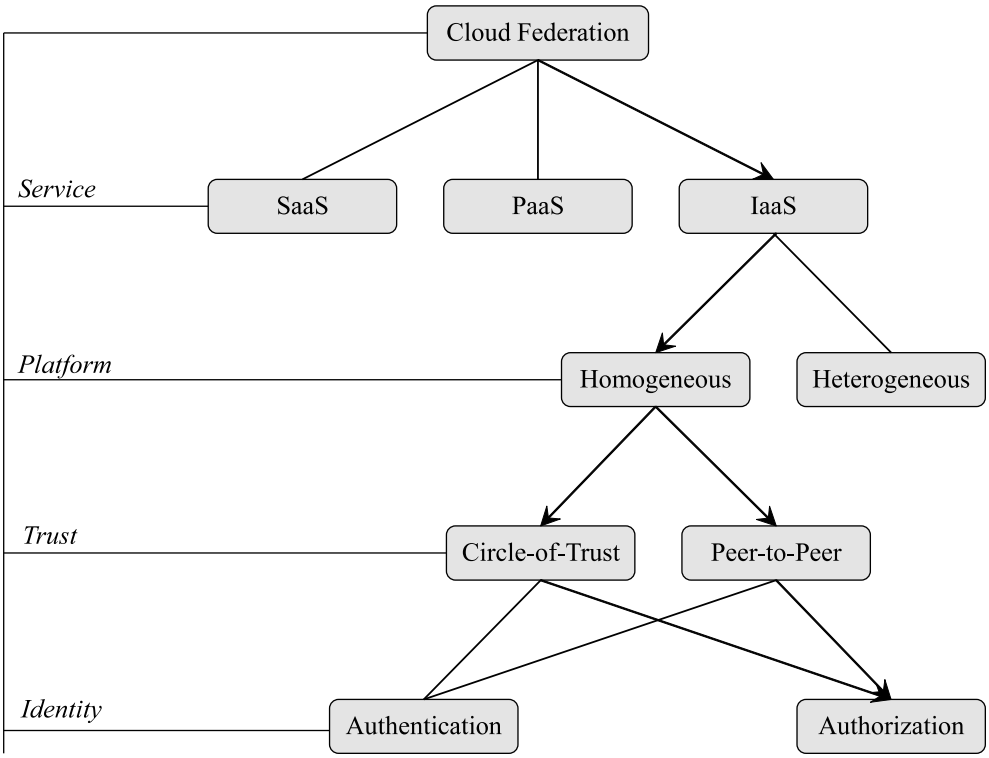


Figure 3.12: Scope of Contribution with Cloud Federation Characteristics.

in section 3.2), we enable collaboration with defined tenant-trusts. In this dissertation, trust is defined as tenant-trust between tenant pairs in Peer-to-Peer and across a group of tenants in Circle-of-Trust.

Our contributions are categorized into Peer-to-Peer and Circle-of-Trust federation models in the following Chapters. In Figure 3.13, we depict a taxonomy of our contributions. Multi-cloud multi-tenant role-based access control model (MC MT-RBAC) enables Peer-to-Peer federation in homogeneous multi-cloud environments using user-role assignments. This model is implemented in OpenStack, an open-source and well accepted cloud platform. Administration operations and implementation considerations are described in Chapter 4 with tenant-trust. In MC MT-RBAC, we realized MT-RBAC [59] in OpenStack with tenant-trust types α, β, γ, and δ where by establishing trust with a tenant from a trusted cloud, user-role assignments across cloud platforms are enabled.

Further, we consider attribute-based model to enable Peer-to-Peer federation in multi-tenant cloud IaaS in Chapter 5. Multi-tenant attribute-based access control model enables Peer-to-Peer

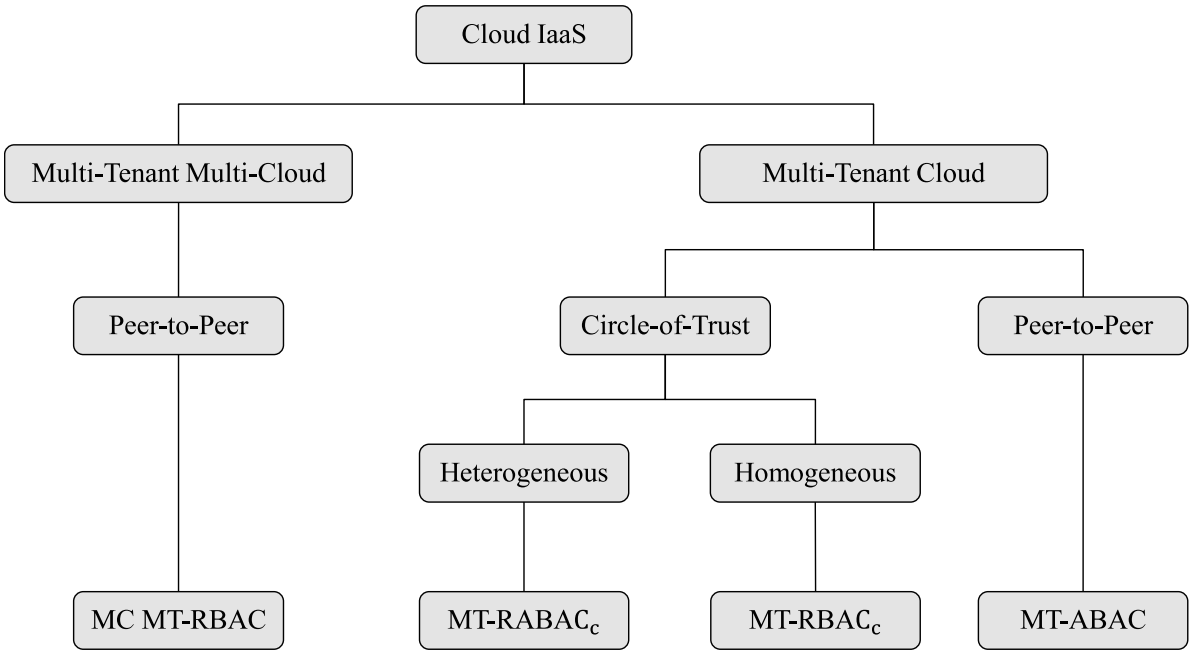


Figure 3.13: Contributions Taxonomy.

federation with attribute assignment tenant-trust. It defines four attribute-based tenant-trust relationships, α, β, γ, and δ where federation is enabled by assigning attributes to trusted tenant users. Type-α enables trustor to assign its user attributes to trustee tenants' users. In type-β, trustee is authorized to assign its user attributes to trusters' users and type-γ enabled trustee to assign trustor user attributes' to its users. Type-δ delegates attribute assignment to trustee in trustor's tenant. MT-ABAC is formalized and further, it is demonstrated that MT-ABAC can be configured to enforce MT-RBAC thus subsuming it as a special case.

Circle-of-Trust federation is discussed in Chapter 6. In Chapter 6, we identify two types of circles, homogeneous circle with unform tenant types and heterogeneous circle with non-uniform tenant types. A multi-tenant role-based access control model in circle (MT-RBAC$_c$) is demonstrated enabling collaboration between tenants in the circle with tenant-trust. In Circle-of-Trust we enable federation by user-role assignments across tenants in the circle. Two tenant-trust types are defined, ϵ and ζ. In type-ϵ user-owner tenants can assign their users to public roles in the circle. In order to protect tenants' resources, we define two types of roles, private and public roles with limited role hierarchy (see section 6.2.1). Cross-tenant user-role assignments in the circle are only

allowed as user to public role assignments. In type-ζ resource owners allowed to assign users in the circle to their public roles. Moreover, we define multi-tenant role-centric attribute-based access control model in circle (MT-RABAC$_c$) to enable federation in heterogeneous circles. Using attributes, we differentiate tenants based on their type, limiting user-role assignments in the circle by tenant attributes.

Chapter 4: PEER-TO-PEER MULTI-CLOUD MT-RBAC MODEL AND OPENSTACK IMPLEMENTATION

In this chapter, we propose a Multi-cloud multi-tenant role-based access control (Multi-cloud MT-RBAC) model providing Peer-to-Peer federation in multi-cloud IaaS. Multi-cloud MT-RBAC defines tenant-trust between tenants in distinct OpenStack cloud platforms. It provides federated user-assignments with defined trust between OpenStack tenants. Moreover, multi-cloud MT-RBAC administration model and implementation considerations in OpenStack identity service is demonstrated in this chapter.

4.1 Multi-Cloud Motivation

Cloud federation is a promising mechanism to share resources across multiple public or private clouds in order to fulfill demanding IT portfolio of enterprises. Cloud provider lock-in is one main drawback of cloud adoption which can be avoided by federated cloud model. Federation allows use of multiple cloud providers to share data and services while data and applications reside in distinct clouds with different security or privacy measures. Federate cloud model offers greater resource pooling, flexibility and dynamicity for organizations. In this scenario, multiple collaborating organizations aim to share resources located across multiple cloud service providers. OpenStack [48] and Amazon Web Services (AWS) [1] as leading cloud platforms currently support federation; however, federation model supported is limited to cloud federation where tenant administrators cannot establish trust with tenants in trusted clouds and administer cross-tenant access .

In this chapter, we present a fine-grained mechanism to establish trust between domains across clouds and enable domain administrators to manage trust and user-role assignments. We use domain and tenant interchangeably since domain is representing tenant in scope of OpenStack and our Peer-to-Peer federation model is implemented in OpenStack. Our contribution is scoped within federation in homogeneous cloud IaaS platforms.

To motivate the problem, we use an enterprise such as Acme in figure 4.1 which stores it's

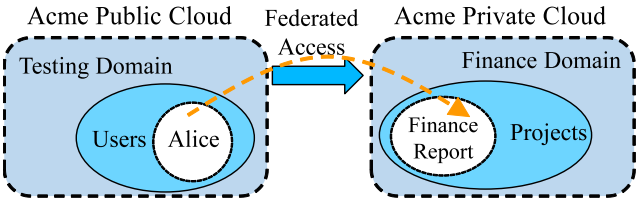


Figure 4.1: Cross-Cloud Domain Federation.

financial data in Acme's private cloud and development applications in Acme's public cloud. Finance domain hosts finance projects in private cloud while Testing domain in public cloud hosts software developer users. Alice as a software developer is working on reports which finance data access is necessary. For security and privacy reasons financial data should not be transferred to other domains. Meanwhile, for administrative reasons it is impractical to provision or assign other domains' users permanently in Finance domain. The practical approach is for Testing domain administrator to establish a relation between two domains in which Finance domain administrator is authorized to assign Alice to Finance report projects. Upon task completion Finance domain administrator can remove user assignments. Testing domain administrators can remove federated relation with Finance domain at any time.

4.2 Role-Based Peer-to-Peer Domain-Trust

This section introduces domain-trust in the concept of Peer-to-Peer federation model in Multi-cloud IaaS. In multi-cloud IaaS, we scope to homogeneous cloud platforms (OpenStack). We defined two type of multi-cloud administration domains in section 3.2, domain-trust provides federation in tenant administrative domain. Trust determines how clouds interact with each other, including which and how much information they share in a trust relationship. We identified domain-trust (tenant-trust) with initiation, direction, and transitivity properties in Peer-to-Peer federation model described in section 3.3. We elaborate domain-trust as unilateral, unidirectional, and non-transitive relationships illustrated in Figure 4.2. We introduced four potential types of trust relations to establish and control cross-domain access in Peer-to-Peer federation model in section 3.4, domain-trust types α, β, γ. and δ.

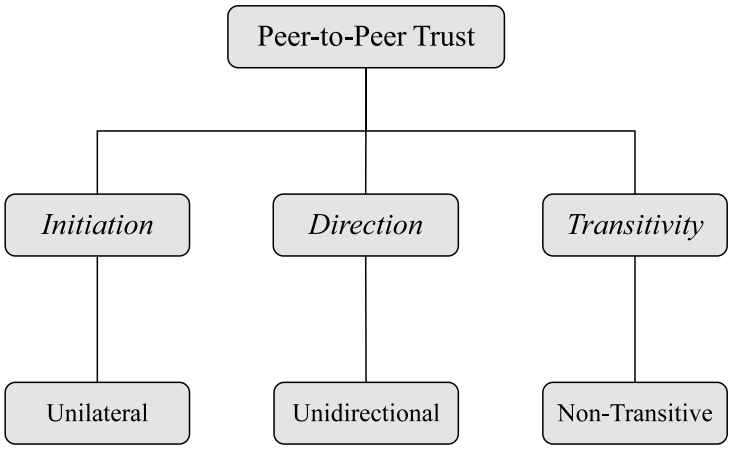


Figure 4.2: A Tree Structure Characterizes Trust.

Introduced domain-trust, provides user-role assignments in Peer-to-Peer federation model where a pair of domains across two distinct cloud platforms establish trust. In the context of domain-trust, each trust type allows different trust relation and cross-domain assignment management. In the following, we use domain A and B in distinct clouds where each has a set of users and resources and cross-domain assignments are $users \rightarrow resources$. We use "$\trianglelefteq$" as a trust relation notion where $A \trianglelefteq B$, states that A trusts B. domain A is trustor and B is trustee domain.

Type-α. Trustor grants inter-cloud access to trustee. It is the most intuitive trust where by trusting a cloud, a trustor cloud domain shares certain resources with trusted cloud domain. *If $A \trianglelefteq_{\alpha} B$, domain A is authorized to assign B's users to domain A's resources. In such trust type, domain A administrator role controls trust relation existence and cross-domain assignments.* Type-α trust is useful when domain A is a resource provider and domain B is an identity provider ($User_B \rightarrow Resource_A$).

Type-β. Trustee grants inter-cloud access to trustor. *If $A \trianglelefteq_{\beta} B$, domain B is authorized to assign domain A's users to its resources. In such trust type, domain A administrator controls trust relation and domain B administrator controls cross-domain assignments.* Domain A must trust domain B with exposing its user set in order to gain cross-domain access. In addition domain A must trust domain B judgment on cross-domain assignments ($User_A \rightarrow Resource_B$).

Type-γ. Trustee takes inter-cloud access to trustor. *If $A \trianglelefteq_{\gamma} B$, domain B is authorized to*

assign its users to domain A's resources. In such trust type, domain A administrator controls trust relation and domain B administrator controls cross-domain assignments. Domain A exposes its selected resources to share with trusted domain B where it trusts domain B judgment on cross-domain assignments ($User_B \rightarrow Resource_A$).

Type-δ. Trustee controls intra-cloud access within trustor. *If $A \trianglelefteq_\delta B$, domain B is authorized to assign domain A's users to resources in domain A. In such trust type, domain A administrator controls trust relation and domain B administrator controls intra-cloud assignments within domain A.* Domain A exposes part or entire set of users and resources to domain B ($User_A \rightarrow Resource_A$). Type-$\delta$ domain-trust enables domains to delegate domain adminstration, useful in organizations with multiple domains across cloud platforms.

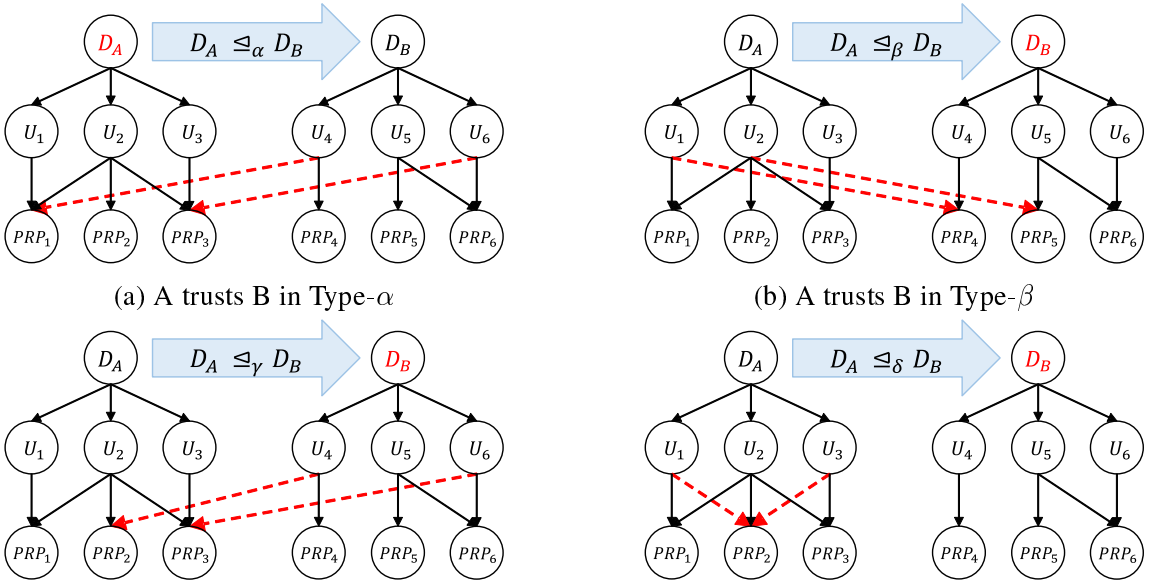
4.2.1 Cross Domain Trust with OpenStack

In our multi-cloud domain-trust, we enable user assignment to shared projects between two clouds upon the trust relation among trusted domains. In each domain a set of resources are shared, more specifically projects. Our model is suited for OpenStack cloud platform, thus we use project-role-pairs (PRP) to represent resources in domains. Roles are cloud global and assigned to users as PRPs. In multi-cloud MT-RBAC, users are assigned to cross-domain PRPs by domain-trust types defined earlier.

Defined domain-trust types $\alpha, \beta, \gamma,$ and δ are used to authorize cross-domain assignments within homogeneous multi-cloud Peer-to-Peer federation. Type-α is illustrated in Figure 4.3a enabling user assignments between an identity provider's users and cloud's PRPs. It allows a domain such as $domain_A$ to share its PRPs by trusting an IdP such as IdP_B ($domain_A \trianglelefteq_\alpha IdP_B$). $domain_A$ is authorized to establish trust relation and control $domain_B$'s user assignments to its PRPs.

Type-β is practical for sharing resources with the purpose of protecting resources' privacy. In Figure 4.3b, $domain_A \trianglelefteq_\beta domain_B$ denotes that $domain_A$ agrees to share its user set with $domain_B$. In such a trust relation $domain_B$ enforce cross-domain user assignments.

Type-γ is useful to share projects within a collaboration group of clouds by extending Peer-to-



(a) A trusts B in Type-α

(b) A trusts B in Type-β

(c) A trusts B in Type-γ

(d) A trusts B in Type-δ

Figure 4.3: Cross-Domain Trust User Assignments

Peer federation into multiple domain-trust type-γ relationships between a group of domains across clouds. Figure 4.3c depicts that trustee $domain_B$ enforces user assignments to $domain_A$'s PRPs.

Type-δ in Figure 4.3d enables trusted cloud ($domain_B$) to administer intra-cloud assignments in trustor $domain_A$. Such trust type is useful to achieve administration federation in a multi-cloud environment.

Using defined domain-trust relationships, we can virtually merge administration of a pair of domains across distinct cloud platforms. We define two types of administration merge across Peer-to-Peer federated domains. Standard merge where Both domain administrators can assign users to federated resources and full merge where both domain administrators have similar authorizations across inter-domain and intra-domain assignments over federated domains. We define standard merge as follows.

Definition 1. *If $domain_A$ trusts $domain_B$ in type β and γ and $domain_B$ trusts $domain_A$ in type β and γ then,*

- *$domain_A$ users can be assigned to $domain_B$ project-role-pairs, both by $domain_A$ administrator and $domain_B$ administrator.*

- *$domain_B$ users can be assigned to $domain_A$ project role pairs both by $domain_A$ administrator and $domain_B$ administrator.*

Also, simply we can fully merge two domains across distinct clouds using domain-trust types β, γ, and δ. We define full merge as follows

Definition 2. *If $domain_A$ trusts $domain_B$ in type β, γ, and δ and $domain_B$ trusts $domain_A$ in type β, γ, and δ then,*

- *$domain_A$ users can be assigned to $domain_B$ project-role-pairs, both by $domain_A$ administrator and $domain_B$ administrator and $domain_A$ administrator can modify user assignments in $domain_B$.*

- *$domain_B$ users can be assigned to $domain_A$ project role pairs both by $domain_A$ administrator and $domain_B$ administrator and $domain_B$ administrator can modify user assignments in $domain_A$.*

4.2.2 Multi-cloud MT-RBAC Administrative Model

In this section, we formalize the multi-cloud Peer-to-Peer domain-trust model introduced in section 4.2.1. Semantics to establish and disband domain-trust relationships of type α, β, γ, and δ are discussed in addition to assignment and unassignment operations for each trust type. Establishing domain-trust is similar between different domain-trust types, thus we describe it once. In the following tables U is the set of users, D is the set domains in the cloud, R is the set of roles, and P is the set of projects.

Establish, An administrator (cloud or domain) user establish trust to another domain in trusted cloud. $Establish(u_1, d_1)$ establishes domain-trust between u_1 current domain and d_1 domain with corresponding type α, β, γ, and δ. Authorization for $Establish$ requires that trustor user have admin role in its domain and a prior coarse-grained trust exists between the domains' clouds. Trusted clouds represented by CT a set of trusted clouds by user u_1 cloud. If the user has admin role and

the trustor domain belongs to trusted clouds then, domain-trust DT is updated with corresponding trust relationship.

Table 4.1 defines domain-trust type-α administrative operations. In $Assignment_{\boldsymbol{\alpha}}(u_1, u_2, p_1, r_1)$, an admin user u_1 assigns a user u_2 to a project p_1 with a role r_1. Authorization requirements are u_1, p_1 and r_1 must belong to the same trustor domain for operation to succeed. Subsequently, UA in trustee's domain is updated with (u_2, p_1, r_1) assignment. User assignment set is represented as UA. In $Unassignment_{\boldsymbol{\alpha}}(u_1, u_2, p_1, r_1)$, similar authorization requirements to $Assignment_{\boldsymbol{\alpha}}$ applies. Thus, (u_2, p_1, r_1) is removed from trustee's UA set. In type-α, $Disband_{\boldsymbol{\alpha}}(u_1, d_1)$ removes domain-trust relation, initially all user-assignments $(project\_owner(p), user\_owner(u))$ which is $(trustor\_domain, trustee\_domain)$ should be removed from trustee's UA set, then domain-trust relationship is deleted.

Domain-trust administrative operations type-β is elaborated in Table 4.2. In $Assignment_{\boldsymbol{\beta}}$, A domain admin user u_1 is allowed to assign trustor domain's user u_2 to its PRP (p_1, r_1). Subsequently, user assignment set UA in trustor domain is updated to reflect that trustor user u_2 is assigned to trustee PRP (p_1, r_1). In $Unassignment_{\boldsymbol{\beta}}$, similar to $Assignment_{\boldsymbol{\beta}}$ u_2 must belong to trustor domain and (p_1, r_1) owns by trustee domain. Afterwards UA is updated by removing (u_2, p_1, r_1) from UA. $Disband_{\boldsymbol{\beta}}$ defines the semantics for removing type-β trust. To that end, first we remove all assignments from trustee domain admin u_1 in trustor's domain UA set, then we remove trust relation $(user\_owner(u_1), d_1)$ between two domain from domain trust set DT.

Type-γ administrative operations are depicted in Table 4.3. In $Assignment_{\boldsymbol{\gamma}}(u_1, u_2, p_1, r_1)$, an admin user u_1 from trustee is authorized to assign it's user u_2 to trustor's (p_1, r_1). Authorization requirements mandate u_1 and u_2 owns by same trustee domain in the domain-trust relationship. Afterwards UA in trustee's domain is updated with (u_2, p_1, r_1) assignment. $Unassignment_{\boldsymbol{\gamma}}$ operation is similar to assignment, however assignment by user admin u_1 should be removed from user assignment set UA in trustee's domain. To disband type-γ trust relation, it is mandatory to remove all user assignments $(project\_owner(p), user\_owner\ (u))$ from trustee's UA set before removing trust.

Table 4.1: Domain-Trust Type-α Administrative Model

Operations	Authorization ($\rightarrow$)	Updates
$\forall u_1 \in U, \forall d_1 \in D$, **Establish**$_{\boldsymbol{\alpha}}(u_1, d_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_cloud(u_1), domain\_owner(d_1)) \in CT$	$DT_\alpha^{'} = DT_\alpha \cup (user\_owner(u_1), d_1)$
$\forall u_1, u_2 \in U, \forall p_1 \in P, \forall r_1 \in R$, **Assignment**$_{\boldsymbol{\alpha}}(u_1, u_2, p_1, r_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_owner(u_1) = project\_owner(p_1)) \wedge (user\_owner(u_1), user\_owner(u_2)) \in DT_\alpha$	$UA_\alpha^{'} = UA_\alpha \cup (u_2, p_1, r_1)$
$\forall u_1, u_2 \in U, \forall p_1 \in P, \forall r_1 \in R$, **Unassignment**$_{\boldsymbol{\alpha}}(u_1, u_2, p_1, r_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_owner(u_1) = project\_owner(p_1)) \wedge (user\_owner(u_1), user\_owner(u_2)) \in DT_\alpha$	$UA_\alpha^{'} = UA_\alpha - (u_2, p_1, r_1)$
$\forall u_1 \in U, \forall d_1 \in D$, **Disband**$_{\boldsymbol{\alpha}}(u_1, d_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_owner(u_1), d_1) \in DT_\alpha$	$UA_\alpha^{'} = UA_\alpha - \{(u, p, r) \mid u \in U, p \in P, r \in R, (project\_owner(p), user\_owner(u)) \in DT_\alpha\}$, $DT_\alpha^{'} = DT_\alpha - (user\_owner(u_1), d_1)$

Table 4.2: Domain-Trust Type-β Administrative Model

Operations	Authorization ($\rightarrow$)	Updates
$\forall u_1 \in U, \forall d_1 \in D$, **Establish**$_{\boldsymbol{\beta}}(u_1, d_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_cloud(u_1), domain\_owner(d_1)) \in CT$	$DT_\beta^{'} = DT_\beta \cup (user\_owner(u_1), d_1)$
$\forall u_1, u_2 \in U, \forall p_1 \in P, \forall r_1 \in R$, **Assignment**$_{\boldsymbol{\beta}}$ (u_1, u_2, p_1, r_1)	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_owner(u_1) = project\_owner(p_1)) \wedge (user\_owner(u_2), user\_owner(u_1)) \in DT_\beta$	$UA_\beta^{'} = UA_\beta \cup (u_2, p_1, r_1)$
$\forall u_1, u_2 \in U, \forall p_1 \in P, \forall r_1 \in R$, **Unassignment**$_{\boldsymbol{\beta}}$ (u_1, u_2, p_1, r_1)	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_owner(u_1) = project\_owner(p_1)) \wedge (user\_owner(u_2), user\_owner(u_1)) \in DT_\beta$	$UA_\beta^{'} = UA_\beta - (u_2, p_1, r_1)$
$\forall u_1 \in U, \forall d_1 \in D$, **Disband**$_{\boldsymbol{\beta}}(u_1, d_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_owner(u_1), d_1) \in DT_\beta$	$UA_\beta^{'} = UA_\beta - \{(u, p, r) \mid u \in U, p \in P, r \in R, (user\_owner(u), project\_owner(p)) \in DT_\beta\}, DT_\beta^{'} = DT_\beta - (user\_owner(u_1), d_1)$

Table 4.3: Domain-Trust Type-γ Administrative Model

Operations	Authorization ($\rightarrow$)	Updates
$\forall u_1 \in U, \forall d_1 \in D$, **Establish**$_\gamma(u_1, d_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_cloud(u_1), domain\_owner(d_1)) \in CT$	$DT_\gamma^{'} = DT_\gamma \cup (user\_owner(u_1), d_1)$
$\forall u_1, u_2 \in U, \forall p_1 \in P, \forall r_1 \in R$, **Assignment**$_\gamma(u_1, u_2, p_1, r_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_owner(u_1) = user\_owner(u_2)) \wedge (project\_owner(p_1), user\_owner(u_1)) \in DT_\gamma$	$UA_\gamma^{'} = UA_\gamma \cup (u_2, p_1, r_1)$
$\forall u_1, u_2 \in U, \forall p_1 \in P, \forall r_1 \in R$, **Unassignment**$_\gamma(u_1, u_2, p_1, r_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_owner(u_1) = user\_owner(u_2)) \wedge (project\_owner(p_1), user\_owner(u_1)) \in DT_\gamma$	$UA_\gamma^{'} = UA_\gamma - (u_2, p_1, r_1)$
$\forall u_1 \in U, \forall d_1 \in D$, **Disband**$_\gamma(u_1, d_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_owner(u_1), d_1) \in DT_\gamma$	$UA_\gamma^{'} = UA_\gamma - \{(u, p, r) \mid u \in U, p \in P, r \in R, (project\_owner(p), user\_owner(u)) \in DT_\gamma\}$, $DT_\gamma^{'} = DT_\gamma - (user\_owner(u_1), d_1)$

Table 4.4: Domain-Trust Type-δ Administrative Model

Operations	Authorization ($\rightarrow$)	Updates
$\forall u_1 \in U, \forall d_1 \in D,$ **Establish**$_{\boldsymbol{\delta}}(u_1, d_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_cloud(u_1), domain\_owner(d_1)) \in CT$	$DT_\delta^{'} = DT_\delta \cup (user\_owner(u_1), d_1)$
$\forall u_1, u_2 \in U, \forall p_1 \in P, \forall r_1 \in R,$ **Assignment**$_{\boldsymbol{\delta}}(u_1, u_2, p_1, r_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_owner(u_2) = project\_owner(p_1)) \wedge (user\_owner(u_1), user\_owner(u_2)) \in DT_\delta$	$UA_\delta^{'} = UA_\delta \cup (u_2, p_1, r_1)$
$\forall u_1, u_2 \in U, \forall p_1 \in P, \forall r_1 \in R,$ **Unassignment**$_{\boldsymbol{\delta}}(u_1, u_2, p_1, r_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_owner(u_2) = project\_owner(p_1)) \wedge (user\_owner(u_1), user\_owner(u_2)) \in DT_\delta$	$UA_\delta^{'} = UA_\delta - (u_2, p_1, r_1)$
$\forall u_1 \in U, \forall d_1 \in D,$ **Disband**$_{\boldsymbol{\delta}}(u_1, d_1)$	$(domain\_admin(u_1) \vee cloud\_admin(u_1)) \wedge (user\_owner(u_1), d_1) \in DT_\delta$	$UA_\delta^{'} = UA_\delta - \{(u, p, r) \mid u, u_1 \in U, p \in P, r \in R, (user\_owner(u_1), user\_owner(u)) \in DT_\delta\}$, $DT_\delta^{'} = DT_\delta - (user\_owner(u_1), d_1)$

Type-δ administrative model is shown in table 4.4. $Assignment_{\boldsymbol{\delta}}$ depicts an intra-domain user assignment by trustee cloud or domain admin user u_1, assigning trustor user u_2 to trustor PRP (p_1, r_1). For $Assignment_{\boldsymbol{\delta}}$ and $Unassignment_{\boldsymbol{\delta}}$, it is required that u_2 and p_1 domains are same trustor domain. In $Disband_{\boldsymbol{\delta}}$, first we remove all intra-domain assignments in trustor domain , then we remove trust relation from DT_δ.

4.3 Implementation

This section reviews multi-cloud MT-RBAC implementation in OpenStack Cloud platform with relevant background on Current federation APIs in OpenStack. We implemented MT-RBAC in OpenStack cloud and multi-cloud MT-RBAC in federated OpenStack cloud platforms providing fine-grained cross-cloud user-assignments based on domain-trust defined in section 4.2.1 with administrative model elaborated in section 4.2.2.

4.3.1 OpenStack Background

OpenStack is an open-source cloud IaaS platform consisting of RESTful API services such as Nova (Compute), Keystone (identity), Neutron (networking), and so on. Keystone authenticates and authorize users to access service in OpenStack. OpenStack access control model consists of entities such as users, groups, projects, domains, and roles. Users are individuals authenticated to Keystone while each group is a set of users. Projects define a container of cloud resources such as virtual machines, storage, and etc. Domain is an administrative boundary which owns users, groups, and projects. Each cloud consist of multiple domains representing an organization (in public clouds), a department of an organization, or an individual who uses cloud services. Roles are global within a cloud boundary. Users and groups are both assigned to roles within project or domain scope. Currently Keystone supports cloud admin and domain admin roles whereas domain admin can only administer within it's home domain. Domain administrators can assign users from other domains to project-role-pairs in their domain.

As of Kilo release OpenStack federation API supports SAML assertions [30] (Keystone generates and consumes assertions). Specifically Keystone to Keystone federation is supported which it allows authenticated users (Keystone as identity provider) to swap their token for a SAML assertion. This assertion is redirected to another Keystone (Keystone as service provider) to get a token back from the second Keystone. Federated user is then mapped to a user or a group from second Keystone and based on assigned roles, it can request a project scoped token. Afterwards the

federated user can use the token to accesses services in the federated OpenStack since the token is valid (usually two hours is the default value).

Currently in Keystone, only cloud administrator can establish trust between two clouds by adding trusted cloud as identity provider or service provider. Identity provider clouds are cloud providers Keystone accepts their assertions and service provider clouds are service providers in which users can access through federation APIs. Currently, domain-trust is not supported in Keystone between domains in a cloud or across multiple clouds. In our implementation, we elaborate domain-trust in OpenStack in addition to cloud trust between clouds.

4.3.2 Multi-Cloud OpenStack Model

In multi-cloud Peer-to-Peer federation model, cross-domain access is granted based on domain-trust between two domains across distinct clouds. In addition to OSAC model [60], federation relation is represented by trustor clouds (identity provider) and trustee clouds (service provider) which is a many-to-one relation. Identity providers are set of clouds which trust to federate their users to the current cloud. Similarly service providers are set of clouds in which current cloud trust to federate its users to their resources. For example in figure 4.1 for Acme Private Cloud, Public Cloud is an identity provider and in Acme Public Cloud, Private Cloud is a service provider. In Multi-cloud OpenStack Access control model (MC-OSAC), administrative model consists of two levels of administrative roles: *cloud-admin* and *domain admin*. *Cloud-admin* refers to cloud-level administrators managing all cloud identity service components. *Domain-admin* is administrator role at domain-level which manages components within its associated domain. We define domain-trust in MC-OSAC as a many-to-many relation between domains in federated clouds. In MC-OSAC we enable domain-trust by remote assignment which is administered by *domain-admin*. Mapping rules define a set of accepted remote users or groups to local domain users and groups. Federated users are mapped to local users and groups by remote mapping in a many-to-one relation as it is depicted in figure 4.4.

[1]It is slightly different from OSAC model where Tokens, PRMS, and Services are omitted due to our scope of contribution.

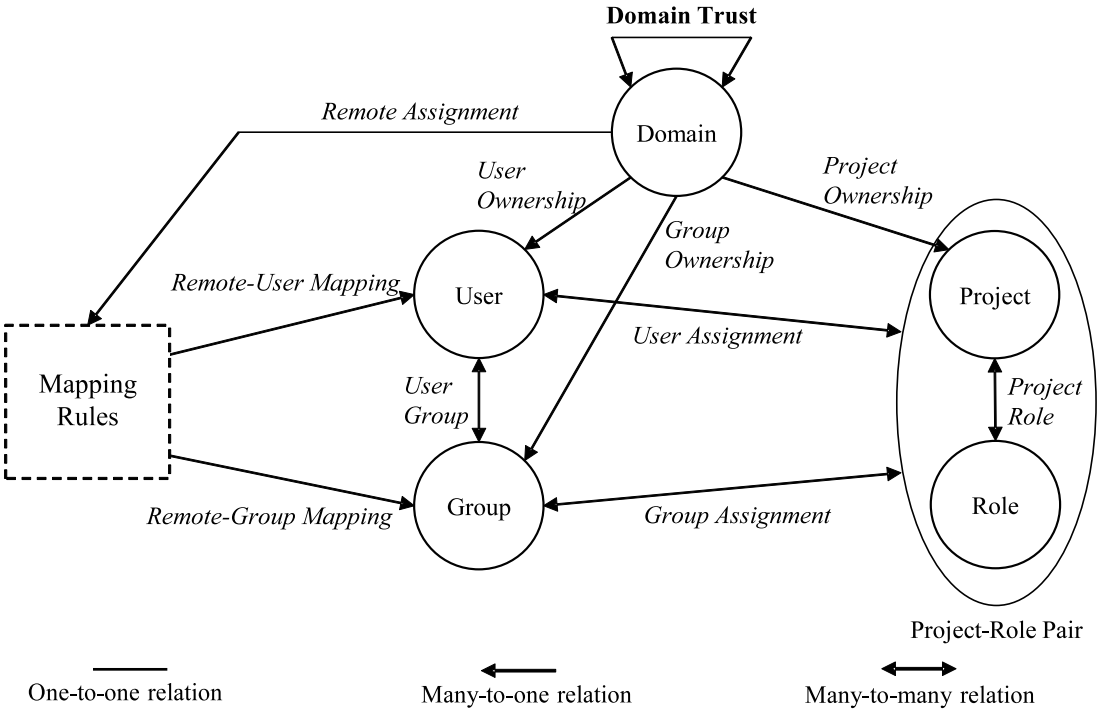


Figure 4.4: Multi Cloud OpenStack Access Control Model (MC-OSAC) with Domain-Trust.[1]

4.3.3 OpenStack Implementation

In this section, implementation considerations are discussed. Keystone as authentication and authorization service is organized as a group of internal services exposed on one or many endpoints. Such internal services are identity, assignment, resource, token, policy, catalog, and federation. In order to deploy our model in OpenStack platform, we modified assignment, policy, and federation internal services in Keystone. We assumed each OpenStack cloud belongs to a single organization while domains represent departments of each organization. In the case of collaboration, multiple domains from distinct OpenStack clouds share resources within specified domain-trust in section 4.3.2.

We implemented the model with two OpenStack DevStack instances. Domain-trust is stored in a MySQL as a *federation-domain-trust* table illustrated in Figure 4.5 showing a domain-trust relation. Trust relation is stored as remote-domain name, local-domain id, and trust-type.

In domain-trust type α, γ, and δ *federation-domain-trust* table is located in identity-provider

remote_domain	local_domain	trust_type
Default	default	beta
domain1	default	beta

Figure 4.5: Federation Domain-Trust Table.

(trustor domain cloud) and in domain-trust type-β, we store the *federation-domain-trust* table in the service provider (trustee domain cloud). We denote the trustor domain cloud as identity provider and trustee domain cloud as service provider. In domain-trust type α and β the *domain-admin* that issues cross-domain assignments and *federation-domain-trust* table are located in the same cloud (type-α in identity provider and type-β in service provider), thus *domain-admin* can modify mapping in the federation API locally. However, with domain-trust type γ and δ the *domain-admin* that issues cross-domain assignments and *federation-domain-trust* table are located in different clouds, thus *domain-admin* must modify mapping in the federation API remotely. We demonstrate type-β trust establishment and user assignments as following. Figure 4.6 illustrates the sequence of establishing and assigning users in a federated OpenStack with domain-trust.

1. A *domain-admin* in trustor cloud initiates trust relation by selecting a trusted cloud from service provider list (note that prior to scoping a token with a project, federated users can list domains with an unscoped token in OpenStack federation).

2. *Domain-admin* token is swapped with SAML assertions and redirected to service provider.

3. *Domain-admin* is mapped to *remote-domain-admin* role which is enabled to modify *federated-domain-trust* table, subsequently *domain-admin* can create the trust relation.

In case of trust relation deletion, Keystone removes all mappings matched with remote-domain name. The cross-domain remote assignment establishment steps are as follows.

1. A *domain-admin* in the service provider initiates remote assignment by selecting a trusted-cloud's domain from identity provider list and *federated-domain-trust* table.

2. *Domain-admin* creates remote assignment of trustor-cloud's domain to its user or groups.

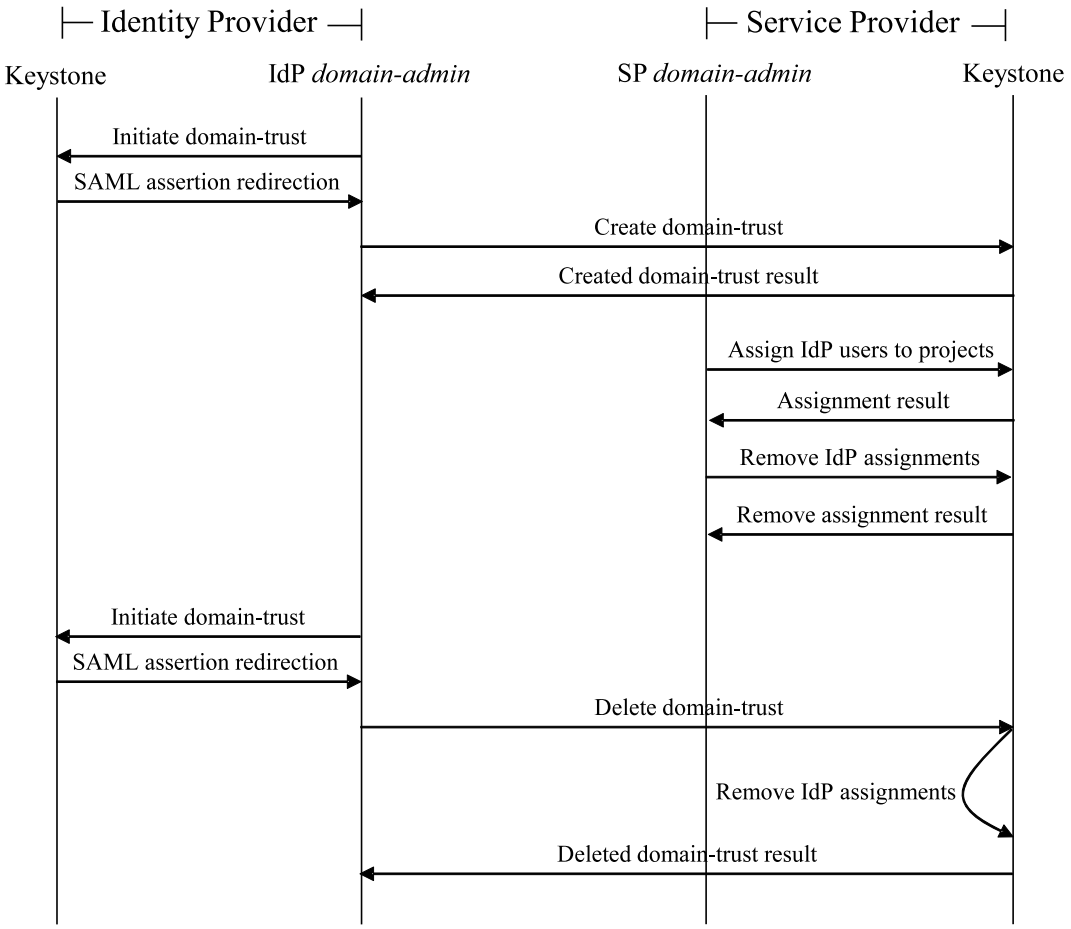


Figure 4.6: Cross-Domain Trust Establishment and Assignment Process.

3. *Federated-domain-trust* table is checked for proper trust relation as well as user or group domain, afterwards remote assignments are created. In case of removing remote assignments, *domain-admin* can only remove mappings it has created.

After remote assignments is created, incoming federated users from identity provider domain are mapped to specified roles based on users and groups it is assigning to in the mapping rules. Within project-role pair permission in OpenStack, federated users can request project scoped token to access authorized resources.

Chapter 5: PEER-TO-PEER MULTI-TENANT ABAC MODEL

This chapter introduces multi-tenant attribute-based access control model (MT-ABAC) providing Peer-to-Peer federation in cloud IaaS. Our approach allows cross-tenant attribute assignment to provide access to shared resources across tenants. We define tenant-trust with attribute assignment. Particularly, our tenant-trust authorizes a tenant to assign its attributes to users from a trusted tenant, enabling access to the trusted tenant's resources. Also, we demonstrate that MT-ABAC can be configured to enforce MT-RBAC as a special case.

5.1 Attribute-Based Peer-to-Peer Motivation

Multi-tenancy is a unique characteristic of cloud computing in which multiple users can utilize shared infrastructure provided by cloud IaaS. Cloud service providers segregate the resources and customer's data into tenants to protect data privacy and integrity. Tenants are isolated containers with tenant-specific virtual computing environments. Each tenant corresponds to an organization, a department of an organization, or an individual who uses cloud services. In this scenario, each tenant is considered as a cloud customer with resources whose integrity and privacy must be protected. The focus on tenant isolation diminishes the scope for multi-tenant federation.

At the dawn of cloud systems, the multi-tenancy concern was resource segregation, whereas recent enterprise cloud adoption has raised the issue of multi-tenancy resource sharing. The drive for multi-tenant federation arises from at least two distinct directions. First, a large organization may utilize multiple tenants for security and reliability, where each tenant can represent a department. For example, an organization's financial department processes sensitive financial data while its marketing department publishes open information to the public, so they need to be isolated but yet may need controlled collaboration. Second, distinct enterprises may have collaborative tasks across their corresponding tenants. Current cloud Infrastructure-as-a-service (IaaS) providers such as Amazon AWS [1] or OpenStack [48] offer limited cross-tenant access [34].

Current cloud IaaS providers such as Amazon or Rackspace provide intra-tenant access control

using variations of the well-known role-based access control (RBAC) [17,54] approach. RBAC has been the dominant access-control paradigm for over two decades. Nevertheless, various limitations of RBAC have been recognized over this period and increasingly there is a push to move towards attribute-based access control [27, 28, 53] in general. ABAC advantages over RBAC specifically in cloud computing have been discussed in the literature [14]. A user's access to a resource in ABAC depends on the relative values of the user and resource attributes. An attribute is simply a name:value pair. Attributes are used to represent security-relevant properties of users and resources. We anticipate that CSPs will incorporate ABAC features in addition to their currently implemented RBAC.

In this chapter we present a novel attribute-based access control model to enable Peer-to-Peer federation between tenants in cloud systems. Our scope is limited to cross-tenant federation in cloud IaaS.

5.2 Attribute-Based Access Control Model ($ABAC_0$)

In this section, we present our core ABAC model which we designate as $ABAC_0$. This model is designed to be sufficient for our purpose in developing MT-ABAC and is not intended to be a comprehensive ABAC model. ABAC has been defined in various ways in the literature, usually for some specific purpose. Our model is specifically motivated by the previously defined $ABAC_\alpha$ model [31] and is compatible with the recently defined NIST ABAC framework [27].

Core $ABAC_0$ model element sets and functions are illustrated in Figure 5.1, which includes three basic components: users (U), objects (O), and actions (A). Attributes are properties associated with users and objects which we represent by $UATT$ and $OATT$ respectively. Users and objects are collectively called entities. Authorization predicates ($Auth$) express access rules in the system which evaluate user attributes against object attributes and render a decision to permit or deny access to the requested resource with respect to the specific action.

Each *attribute* is a function which takes users or objects as input and returns a value from the attribute's range (we use the terms range and scope interchangeably). For example, a user attribute

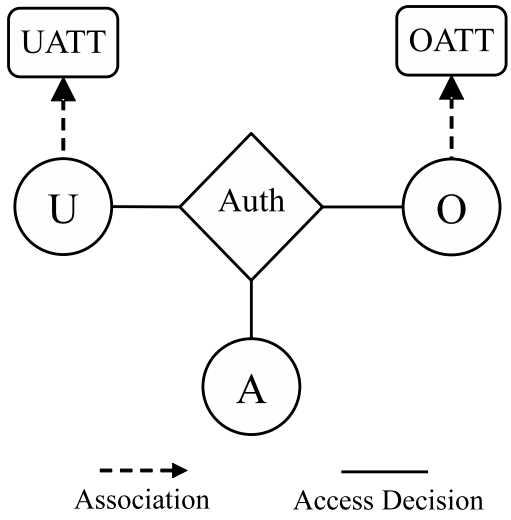


Figure 5.1: Core $ABAC_0$ Model Structure.

function such as $Role \in UATT$ maps $u_1 \in U$ to a value $cloud\_admin$. Depending upon attribute type each attribute function will return a single value or a set of values. An atomic-valued attribute will return one value while a set-valued attribute will return a subset of values within its defined scope.

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

Objects are system resources for which access should be protected such as files, applications, virtual machines (VMs), etc. Objects are associated with attribute functions ($OATT$) representing resource properties such as risk level, location, and classification. At creation or modification time object attributes might be constrained by the attributes of creating user in the system, for example, a new VM object can inherit attributes such as VM owner from corresponding user attributes such as user id. The details of such constraints are not material for our purpose in this dissertaion, hence we do not explicitly model them. The approach of $ABAC_\alpha$ [31] in this regard could be adapted to $ABAC_0$.

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

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

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

Definition 3. *Core ABAC$_0$ is defined by the basic component sets, functions and authorization policy language given below.*

- *U and O represent finite sets of existing users and objects respectively.*
- *A represents a finite set of actions available on objects. Typically $A = \{create, read, update, delete\}$.*
- *$UATT$ and $OATT$ represent finite sets of user and object attribute functions respectively.*
- *For each att in $UATT \cup OATT$, $Scope(att)$ represents the attribute's scope, a finite set of atomic values.*
- *$attType : UATT \cup OATT \rightarrow \{set, atomic\}$, specifies attributes as set or atomic valued.*
- *Each attribute function maps elements in U and O to atomic or set values as follows.*

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

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

- *For each* $a \in A$, $Authorization_a(u : U, o : O)$ *is a propositional logic predicate, defined using the following language:*
  - $\varphi ::= \varphi \wedge \varphi \mid \varphi \vee \varphi \mid (\varphi) \mid \neg\varphi \mid \exists x \in set.\varphi \mid \forall x \in set.\varphi \mid set \triangle set \mid atomic \in set \mid atomic \nabla atomic$
  - $set ::= setuatt(u) \mid setoatt(o)$
  - $atomic ::= atomicuatt(u) \mid atomicoatt(o)$
  - $\triangle ::= \subset \mid = \mid \subseteq \mid \not\subseteq$
  - $\nabla ::= < \mid = \mid \leq$
  - $setuatt \in \{uatt \mid uatt \in UATT \wedge attType(uatt) = set\}$
  - $setoatt \in \{oatt \mid oatt \in OATT \wedge attType(oatt) = set\}$
  - $atomicoatt \in \{oatt \mid oatt \in OATT \wedge attType(oatt) = atomic\}$
  - $atomicuatt \in \{uatt \mid uatt \in UATT \wedge attType(uatt) = atomic\}$

Core $ABAC_0$ is a simplified version of $ABAC_\alpha$ [31], suitable for our purpose in this dissertation. In particular it eliminates subjects as being distinct from users as is in $ABAC_\alpha$, and simply treats them to be equivalent.

5.3 Multi-Tenant $ABAC_0$ Model

In this section we build upon $ABAC_0$ to formulate a multi-tenant attribute-based access control model enabling cross-tenant collaboration which we designate as MT-$ABAC_0$. The model structure is depicted in Figure 5.2, adding the tenant (T) entity in addition to the users and objects of core $ABAC_0$. *Tenants* are isolated operation domains leased by cloud service consumers.

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

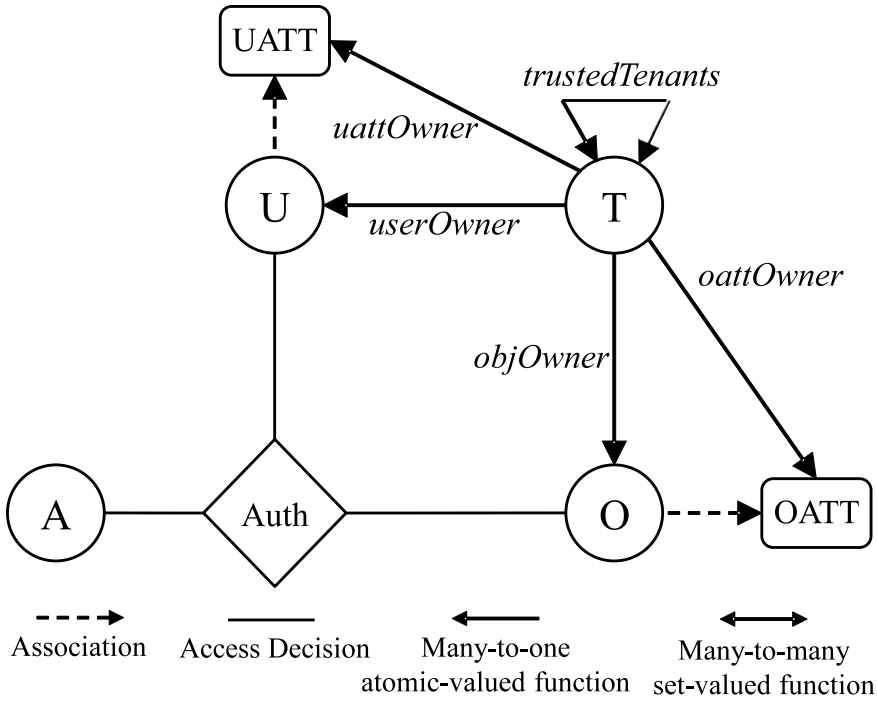


Figure 5.2: Multi-Tenant ABAC$_0$ Model Structure.

each object to have a system defined attribute $objOwner$ which similarly is a many-to-one atomic-valued function from objects O to tenants T.

Further, each user attribute and each object attribute is also uniquely owned by a single tenant, depicted respectively by the many-to-one atomic-valued functions $uattOwner$ and $oattOwner$ in Figure 5.2. The crucial concept is that each tenant is responsible for assigning values to attributes that it owns. With isolated tenants, a user can have assigned values only for those attributes owned by the user's owning tenant. We will see that, with appropriate trust relationship between tenants, users belonging to one tenant can be assigned values for attributes belonging to a different tenant. In our model for objects, we require that an object can have assigned values only for those attributes owned by the object's owning tenant. It is not possible for an object to be assigned values for attributes that belong to a tenant that does not own that object, regardless of tenant trust relationships. In summary cross-tenant attributes can be assigned to users under appropriate trust relationships but not to objects.

We define trust as a required attribute function $trustedTenants$ mapping trustor tenant to

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

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

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

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

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

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

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

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

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

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

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

[1] More generally different kinds of trust could be considered as discussed in Section 5.3.2.

5.3.1 Formal MT-ABAC$_0$ Model

We summarize the above in the following definition.

Definition 5. *Multi-tenant ABAC$_0$ is defined by the following enhancement and modifications to core ABAC$_0$.*

- U, O, *and* A *are defined as in core ABAC$_0$.*
- T *represents a finite set of existing tenants.*
- $UATT$, $OATT$, *Scope, and attType are defined as in core ABAC$_0$.*
- $userOwner : (u : U) \rightarrow T$, *required attribute function mapping user* u *to owner tenant* t.
- $objOwner : (o : O) \rightarrow T$, *required attribute function mapping object* o *to owner tenant* t.
- $MATT = \{uattOwner, oattOwner\}$, *required meta-attribute functions.*
  - $uattOwner : (uatt : UATT) \rightarrow T$, *meta attribute function, mapping user attribute* $uatt$ *to attribute owner tenant* t.
  - $oattOwner : (oatt : OATT) \rightarrow T$, *meta attribute function, mapping object attribute* $oatt$ *to attribute owner tenant* t.
- $trustedTenants : (t : T) \rightarrow 2^T$, *required attribute function, mapping tenant* t *to powerset of trusted* T, *called tenant-trust, written as* $\trianglelefteq$ *where* $t_1 \trianglelefteq t_2$ *iff* $t_2 \in trustedTenants(t_1)$ *(i.e., trustor tenant* t_1 *trusts trustee tenant* t_2*). Trustee tenant* t_2 *can assign its attribute values* $uatt_{t_2}$ *to users* u_{t_1} *from trustor tenant* t_1 *where* $t_2 \in trustedTenants(userOwner(u))$.
- *Each attribute function* $uatt \in UATT$ *is modified to be a partial function.*
  $\forall uatt \in UATT.uatt : U \hookrightarrow \begin{cases} Scope(uatt) & if\ attType(uatt) = atomic \\ 2^{Scope(uatt)} & if\ attType(uatt) = set \end{cases}$
  $uatt(u : U)$ *is defined only if* $(uattOwner(uatt) = userOwner(u)) \vee$
  $(uattOwner(uatt) \in trustedTenants(userOwner(u)))$.

- *Each attribute function* $oatt \in OATT$ *is modified to be a partial function.*

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

  $OATT(o : O)$ *is defined only if oattOwner(oatt) = objOwner(o).*

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

5.3.2 Peer-to-Peer Attribute-Based Tenant-Trust

In a tenant trust relation, in general there are two issues: (i) who controls trust relation's existence, and (ii) who has the authority to issue cross-tenant assignments. Together these characterize the trust type. Moreover, trust relationships we elaborated in this section are unilateral, unidirectional, and non-transitive (see section 3.3 for more detail). In this dissertation, for simplicity, we adopted a specific definition of trust where trustee tenant is authorized to assign its attribute values to trustor tenant's user attributes which is analogous to the type-β tenant-trust of [59]. In this section, we briefly discuss trust types analogous to the type-α and type-γ tenant-trust types.

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

Definition 6. *If $T_A \trianglelefteq_\alpha T_B$, Tenant T_A is authorized to assign values for T_A's user attributes to Tenant T_B's users. Tenant T_A controls tenant-trust existence and cross-tenant attribute assignments.*

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

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

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

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

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

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

Definition 7. *If $T_A \trianglelefteq_\gamma T_B$, Tenant T_B is authorized to assign values for T_A's user attributes to Tenant T_B's users. Tenant T_A controls tenant-trust existence while T_B controls cross-tenant attribute assignments.*

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

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

In this section we first give a definition of multi-tenant RBAC (MT-RBAC$_0$) adapted from various slightly different but related models given in [59, 60, 61]. We then show how MT-RBAC$_0$ can be configured in MT-ABAC$_0$.

5.4.1 Multi-Tenant RBAC$_0$ Model

MT-RBAC$_0$ model element sets and relations are illustrated in Figure 5.3, showing the six components: tenants (T), users (U), roles (R), operations (OPS), objects (OBS), and permissions ($PRMS$). A *user* is an individual which is associated with a single tenant via UO relation. We recognize *role* as a job function associated with a single tenant while a tenant has multiple roles. *Objects* are tenant resources in the system (each object has a single owner tenant) which are cou-

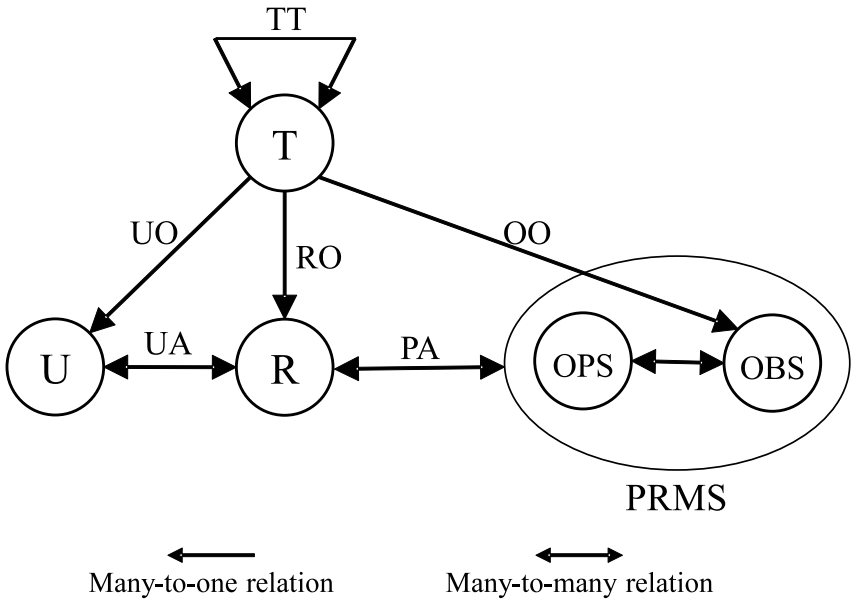


Figure 5.3: Multi-Tenant RBAC$_0$ Model Structure.

pled with *operations*. In RBAC, *permissions* are operation, object pairs indicating operations on objects.

MT-RBAC$_0$ model is defined in terms of users, roles, and objects owned by tenants. These ownership relations are many-to-one representing tenant ownership which is depicted as user-ownership (UO), role-ownership (RO), and object-ownership (OO) in figure 5.3.

As core to RBAC, user assignment (UA) and permission assignment (PA) relations enable assignment of users and permissions to roles. Tenant-trust (TT) identifies a many-to-many trust relation between tenants. Similar to MT-ABAC$_0$ we use $\trianglelefteq$ to show trust between two tenants such T_A and T_B as $T_A \trianglelefteq T_B$ means trustor tenant T_A trusts, trustee tenant T_B. With this specification, we define tenant-trust relation as follows.

Definition 8. *If $T_A \trianglelefteq T_B$, Tenant T_B is authorized to assign Tenant T_A's users to T_B's roles.*

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

With existence of trust between tenants, user assignment is defined as many-to-many user rela-

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

Each user is assigned to one or many roles within their resident tenants or trusted tenants. The function *assigned_user_roles* returns the roles assigned to a user. The permissions available to a user, are permissions assigned to roles (permissions available to a role are expressed by function $assigned\_permissions$) that are available to a user which are given by function *authorized_user_permissions*. Function *authorized_user_permissions* designates set of permissions available to each user in the system.

5.4.2 Formal MT-RBAC$_0$ Model

We formally define MT-RBAC$_0$ as follows.

Definition 9. *Multi-tenant RBAC$_0$.*

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

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

- $owner\_user : (u : USERS) \rightarrow TENANTS$, *the mapping of user* u *into its owner tenant. Formally:* $owner\_user(u) = t$ *where* $(u, t) \in UO$.

- $owner\_role : (r : ROLE) \rightarrow TENANTS$, *the mapping of role* r *into its owner tenant. Formally:* $owner\_role(r) = t$ *where* $(r, t) \in RO$.

- $owner\_object : (ob : OBS) \rightarrow TENANTS$, *the mapping of object* ob *into its owner tenant. Formally:* $owner\_object(ob) = t$ *where* $(ob, t) \in OO$.

- $TT \subseteq TENANTS \times TENANTS$, *is a many-to-many reflexive relation on TENANTS called tenant trust relation, written as* $\trianglelefteq$ *where* $t_1 \trianglelefteq t_2$ *(trustor tenant* t_1 *trusts trustee tenant* t_2*) only if all users of* t_1 *can be assigned to roles of* t_2.

- $trustee\_tenants : (t : TENANTS) \rightarrow 2^{TENANTS}$, *the mapping of tenant* t *into a set of trusted tenants. Formally:* $trustee\_tenant(t) = \{t' \in TENANTS \mid t \trianglelefteq t'\}$.

- $UA \subseteq USERS \times ROLES$, *a many-to-many mapping user-to-role assignment relation requiring that* $(u, r) \in UA \Rightarrow owner\_user(u) = owner\_role(r) \vee owner\_user(u) \trianglelefteq owner\_role(r)$.

- $PA \subseteq PRMS \times ROLES$, *a many-to-many mapping permission-to-role assignment relation requiring that* $((op, ob), r) \in PA \Rightarrow owner\_object(ob) = owner\_role(r)$.

- $assigned\_roles : (op : OPS, ob : OBS) \rightarrow 2^{ROLES}$, *the mapping of object operation pair* (op, ob) *into a set of roles. Formally:* $assigned\_roles(op, ob) = \{r \in ROLES \mid ((op, ob), r) \in PA\}$.

- $assigned\_user\_roles : (u : USERS) \rightarrow 2^{ROLES}$, *the mapping of user* u *into a set of roles. Formally:* $assigned\_user\_roles(u) = \{r \in ROLES \mid (u, r) \in UA\}$.

- $assigned\_permissions : (r : ROLES) \rightarrow 2^{PRMS}$, *the mapping of role* r *into a set of permissions. Formally:* $assigned\_permissions(r) = \{p \in PRMS \mid (p, r) \in PA\}$.

- $authorized\_user\_permissions : (u : USER) \rightarrow 2^{PRMS}$, *the mapping of user* u *into a set of permissions.* $authorized\_user\_permissions(u) = \bigcup_{r \in assigned\_user\_roles(u)} assigned\_permissions(r)$.

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

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

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

5.4.4 Formal MT-RBAC$_0$ Configured in MT-ABAC$_0$

The summary of above is formalized as follows.

Definition 10. *A given MT-RBAC$_0$ instance is configured in MT-ABAC$_0$ as follows.*

- $U = USERS, O = OBS, A = OPS = \{a_1, ..., a_n\}$ *where* $n = |A|$,

and $T = TENANTS = \{t_1, ..., t_m\}$ *where* $m = |T|$.

- $UATT = \{userRole_j \mid j = 1, ..., |T|\}$.
- $OATT = \{objRole_{i,k} \mid i = 1, ..., |A|, k = 1, ..., |T|\}$.
- $userOwner : (u : U) \rightarrow T$, *required attribute function, mapping user* u *to owner tenant* t. ***Formally:*** $userOwner(u) = owner\_user(u)$.
- $objOwner : (o : O) \rightarrow T$, *required attribute function, mapping object* o *to owner tenant* t. ***Formally:*** $objOwner(o) = owner\_object(o)$.
- $userRole_j : (u : U) \rightarrow 2^{ROLES}$ *where* $t_j \in T$, *attribute function, mapping user* u *to powerset of* $ROLES$. ***Formally:*** $userRole_j(u) = \{r \in ROLES \mid r \in assigned\_user\_roles(u) \wedge owner\_role(r) = t_j\}$.
- $objRole_{i,k} : (o : O) \rightarrow 2^{ROLES}$ *where* $a_i \in A$ *and* $t_k \in T$, *attribute function, mapping object* o *for operation* a_i *to powerset of* $ROLES$. ***Formally:*** $objRole_{i,k}(o) = \{r \in ROLES \mid r \in assigned\_roles(a_i, o) \wedge owner\_role(r) = t_k\}$.
- $MATT = \{uattOwner, oattOwner\}$.
  - $uattOwner : (userRole_j : UATT) \rightarrow T$, *meta attribute function, mapping user role attribute* $userRole_j$ *to attribute owner tenant* t_j. ***Formally:*** $uattOwner(userRole_j) = t_j$.
  - $oattOwner : (objRole_{i,k} : OATT) \rightarrow T$, *meta attribute function, mapping object role attribute* $objRole_{i,k}$ *for operation* a_i *to attribute owner tenant* t_k. ***Formally:*** $oattOwner(objRole_{i,k}) = t_k$.
- $trustedTenants : (t : T) \rightarrow 2^T$, *attribute function, mapping tenant* t *to powerset of trusted* T. ***Formally:*** $trustedTenants(t) = trustee\_tenants(t)$.

- $Authorization_i\ (u\ :\ U, o\ :\ O)\ =\ \bigvee_{k=1,\ldots,|T|}[userRole_k(u) \cap objRole_{i,k}(o)\ \neq\ \emptyset \wedge (t_k\ =\ userOwner(u) \vee t_k \in trustedTenants(userOwner(u))]$.

Chapter 6: CIRCLE-OF-TRUST CLOUD MODELS

A Circle-of-Trust established trust relations between each pair of tenants in the circle. This chapter presents a novel extension of role-centric access control models to provide collaboration in the context of homogeneous and heterogeneous circles. In a homogeneous circle tenants can equally assert cross-tenant user assignments to enable access to shared resources. In a heterogeneous circle with non-uniform tenants, tenant attributes are used to distinguish which user-assignments are authorized. we defined two trust types called ϵ and ζ respectively. Our scope in this chapter is federation in homogeneous and heterogeneous multi-tenant circles in cloud IaaS. The extension of the models proposed in this chapter to multi-cloud would be straightforward in scope of homogeneous multi-cloud IaaS.

6.1 Circle of Trust in Cloud

A group of tenants which adhere to a set of common policies, trust relations and collaboration interfaces create a Circle-of-Trust where collaboration interactions are defined between all members. Scenarios such as a large enterprise with multiple tenants collaborating in a public cloud, an organization with tenants across public and private clouds, or tenants from multiple organizations performing collaborative tasks are motivating use cases for circle-of-trust.

In contrast to one-to-one relationships between tenants (peer-to-peer), the circle-of-trust provides an association of tenants where tenants can issue user-assignments (assertions). Circle-of-trusts are distinguished in terms of the following

- **Circle governance**. A circle-of-trust can be governed by a creator, a group of members, or all members. Centralized circles are governed by a single founder governing the rules and contracts in circle. Consortium model defines a set of founders where they set the rules of operation in circle. In collaborative model all partners share the governance of the circle.

- **Partners Selection**. In a circle to select partners either certain pre-established trust agree-

ments must exist prior to selection, or attributes such as type of tenants or certificates must be established.

- **Partners interaction**. Authorized interactions between partners in the circle must be defined. Either equal assertions are allowed (homogeneous) or assertions are limited to tenant attributes such each tenant's security level (heterogeneous).

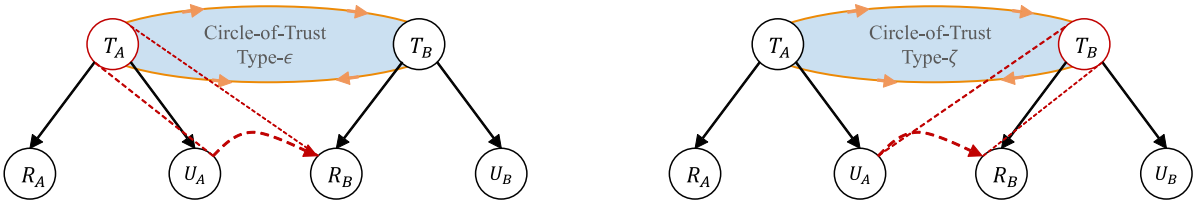
We focus on collaborative governance, pre-established partners, and providing both type of partners' interactions in the circle.

RBAC and its variations have been successfully applied to cloud IaaS providing collaboration within single-cloud or multi-cloud systems. In RBAC access permissions are assigned to roles and roles are assigned to users. RBAC abstracts permissions into roles and role relations. RBAC limitations have been recognized over it extensive use in industry leading to a push towards using attributes with roles [33]. One method of attribute incorporation to roles, is adding attributes to roles as role-centric attributes which takes advantage of roles' simplicity and attributes flexibility. With this consideration, cloud service providers can incorporate ABAC features to their current RBAC models such with role-centric methods to adopt convenience of RBAC with flexibility of ABAC models.

In this chapter we present multi-tenant role-centric models with cross-tenant user-assignments. To our knowledge this is the first work considering role-centric models in circle-of-trust context. We present novel role-based and role-centric attribute-based access control models to enable collaboration in a multi-tenant cloud IaaS circle-of-trust.

6.1.1 Tenant-Trust in Circle

Tenant-trust relationships are elaborated as multilateral, bidirectional, and transitive trust relationship for homogeneous circles and in heterogeneous circles trust relationship is multilateral, unidirectional, and non-transitive (see section 3.3). In a unidirectional trust relationship, common in peer-to-peer, trust is initiated and established between two tenants denoted as trustor and trustee.



(a) Circle-of-Trust Type-ϵ. (b) Circle-of-Trust Type-ζ.

Figure 6.1: Cross-Tenant User Assignment in Circle-of-Trust.

In a trust relation, trustor tenant is willing to trust another tenant denoted as trustee tenant. In our scope, trust is initiated multilaterally between principals in a circle. In the context of circle, trustor and trustee are not distinguished in trust relations between tenants. We identify tenants involve in a cross-tenant assignment as user-owner and resource-owner tenants. User-owner tenant owns the users in the cross-tenant assignment and resource-owner tenant owns the roles to which users are assigned. Central to tenant-trust defined in this dissertation is authorizing user-owner or resource-owner tenant to assert cross-tenant user-role assignments.

We use "$\triangleleft$" to represent tenant-trust where $T_A \triangleleft T_B$ signifies that tenant A trusts tenant B. In this relation, T_A is user-owner tenant and T_B is resource-owner tenant. Regardless of circle entity coupling, we define two types of tenant trust relations labeled as type-ϵ and type-ζ. Each tenant-trust relation type is applied to all tenants in the circle. In type-ϵ circle, user-owner tenants are authorized to assign users to roles in the circle. The following defines type-ϵ tenant-trust illustrated in Figure 6.1a.

Definition 11. *If $T_A \triangleleft_\epsilon T_B$, then tenant T_A is authorized to assign its users to T_B's roles. Tenant T_A controls user assignments.*

In type-ζ circle, resource-owner tenants are authorized to assign users in the circle to their roles. Type-ζ is defined as the following depicted in Figure 6.1b.

Definition 12. *If $T_A \triangleleft_\zeta T_B$, then tenant T_B is authorized to assign T_A's users to its roles. Tenant T_B controls user assignments.*

In homogeneous circles all peers trust each other and trust is transitive therefore $T_A \triangleleft T_B$ if

and only if $T_B \triangleleft T_A$. However, in heterogeneous circles trust relations are unidirectional and non-transitive as a result $T_A \triangleleft T_B$ may not imply $T_B \triangleleft T_A$ or vise versa.

Each tenant-trust type caters to a different security concern and objective in circle-of-trust collaboration. In type-ϵ, the objective is to merely share resources with trusted tenants. Resource-owner tenants can decide on resources are shared within principals in the circle using role hierarchy described in section 6.2. A circle of institutions is an instance of type-ϵ circle where computing resources are shared between institutions. In this situation, each tenant administrator can assign its users to shared resources. For instance, a Circle-of-Trust of universities motivates this type of trust where member universities can assign their students and employees to shared resources within the circle.

In type-ζ circle, shared resources privacy and integrity is the main concern where tenants' resources are highly sensitive data and tenants choose to control access to shared resources. Resource-owner tenants control user assignments where they can assign users in the circle to roles in their tenants to authorize access. A circle of banks carries such sensitivity where banks ought to control access to their shared resources even from trusted banks within the circle. Each bank administrator assign users from tenants in the circle to its roles, enabling access to its shared resources.

6.2 Homogeneous Role-Based Circle-of-Trust

In this section, we formally present a multi-tenant role-based access control model to enable collaboration in a homogeneous circle-of-trust which we refer to as MT-RBAC$_c$. In a homogeneous circle, tenants are equally authorized to make assertions. Collaboration in MT-RBAC$_c$ is issued through cross-tenant user assignments with respect to circle types ϵ and ζ.

6.2.1 Multi-Tenant Role-Based Circle-of-Trust (MT-RBAC$_c$)

MT-RBAC$_c$ model element sets and relations are depicted in Figure 6.2. These includes sets of eight basic elements: tenants (T), users (U), private roles (R_{prv}), public roles (R_{pub}), roles (R), operations (OPS), objects (OBS), and permissions ($PRMS$). In MT-RBAC$_c$ access is funda-

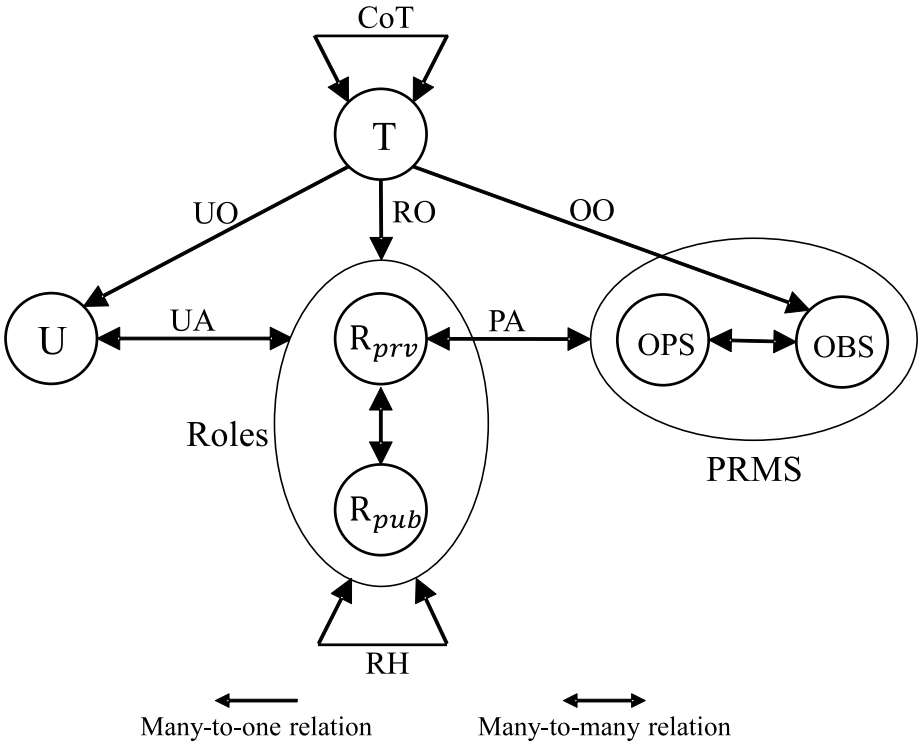


Figure 6.2: Multi-Tenant RBAC Circle-of-Trust.

mentally defined in terms of intra-tenant and cross-tenant. Users are assigned to private roles and permissions in intra-tenant assignments. In cross-tenant assignments, users are assigned to public roles and public roles inherit private roles with limited role hierarchy. MT-RBAC$_c$ includes limited role hierarchy relation utilizing a set of public roles to create a level of abstraction which protects objects from direct assignments in a circle-of-trust.

A *user* is a human, non-person entity, an application, or a process making requests to access objects. We scope users as a human for simplicity where $u \in U$ and U is the global set of users associated to a tenant in the cloud. In our model, a *tenant* is considered as a virtual container with tenant-specific environment for cloud services leased to cloud consumers. Practically, a tenant hosts a project, department, or an organization. Each tenant is represented as $t \in T$ where T is a global set of tenants in the cloud. Tenants are regarded as an administration domain associated to users, roles, and objects. Each user, role, and object belongs to a single tenant, called its owner, while a tenant can own multiple instances of these.

A *role* is a job function in the cloud associated with a tenant. Roles are partitioned to public role and private role disjoint sets, R_{pub} and R_{prv} respectively. A *public role* is accessible by trusted tenants in a circle-of-trust. A *private role* is accessible only within its owner tenant. In contrast to private roles, public roles are not associated with permissions directly.

Resources in a tenant are represented by *objects*, each associated with an owner tenant. In our model, we consider objects as virtual machines, images, storage objects, networking $L2$ or $L3$ services, etc. Objects are owned by a tenant and paired with *operations* comprising the set of actions on cloud resources such as create, read, update, and delete. A *permission* is an authorization to perform an operation on a requested object in a tenant, such as a permission to create virtual machines in a tenant.

Central to MT-RBAC$_c$ model is tenant and role relations. Users, roles, and permissions are global sets in a cloud but they are defined per tenant with a single tenant owner. Tenant ownership relation is fundamental in how we authorize collaboration. Figure 6.2 illustrates *user ownership* (UO), *role ownership* (RO), and *object ownership* (OO) relations. The arrows indicate a many-to-one relation (e.g., a role can be owned by one tenant while a tenant owns many roles) stating the relation between corresponding components and owner tenant.

In Figure 6.2, CoT depicts circle-of-trust relation between tenants. It is defined as a many-to-many relation between tenants. As stated in section 6.1.1, we use "$\lhd$" to show trust relation between tenants. In a homogeneous circle, $T_A \lhd T_B$ signifies that T_A and T_B trust each other in a circle. In a homogeneous circle, each tenant-trust type-ϵ or type-ζ authorizes user-owner or resource-owner tenants. We define CoT$_\epsilon$ as follows.

Definition 13. *In a homogeneous CoT_ϵ, for all tenants t_1 where $t_1 \lhd_\epsilon t_2$, tenant t_1 is authorized to assign its users to public roles in t_2. Tenant t_1 controls cross-tenant user-role assignments of t_1's users to t_2's roles.*

In type-ϵ, user-owner tenants are authorized to assert assignments, while in type-ζ role-owner tenants are authorized. Circle-of-trust type-ζ is defined as follows.

Definition 14. *In a homogeneous CoT_ζ, for all tenants t_1 where $t_1 \lhd_\zeta t_2$, tenant t_2 is authorized to assign users from t_1 to public roles in t_2. Tenant t_2 controls cross-tenant user-role assignments of t_1's users to t_2's roles.*

In homogeneous circles, trust is transitive and tenants are equally authorized, therefore in above definitions $t \lhd_\zeta t'$ if and only if $t' \lhd_\zeta t$.

In MT-RBAC$_c$ the concept of role relations with users and permissions as illustrated in Figure 6.2 with *user assignment* (UA) and *permission assignment* (PA) modified to reflect multi-tenancy. Fundamentally MT-RBAC$_c$ provides collaboration with user-assignment between tenants in a circle-of-trust. To that end, user assignment is defined only if user and role owned by the same tenant or user is owned by trustee tenant in type-ϵ and trustor tenant in type-ζ. User assignment of user u to role r is defined only if

$$(owner\_user(u) = owner\_role(r) \wedge r \in R) \vee$$
$$(owner\_user(u) \lhd owner\_role(r) \wedge r \in R_{pub})$$

Similarly permission assignment is a many-to-many relation which assigns roles to permissions within a tenant. Permission assignment is only defined as an intra-tenant assignment, therefore a permission p is assigned to a role r only if

$$(owner\_role(r) = owner\_object(o) \wedge r \in R_{prv})$$

In order to provide a secure collaboration with respect to tenants' authority on shared resources in a circle, we utilize limited role hierarchy. With two disjoint sets of private and public roles, permissions are available to tenants in the circle only through public roles. This arrangement provides granularity of permission to role assignment and user to role assignment. We define limited *role hierarchy* as a partial order on roles with conditional inheritance. We use "$\succeq$" to show role inheritance relation where a parent role inherits child's role permissions with following conditions.

- Private roles can inherit private roles only if both are owned by the same tenant.

$$(r_1, r_2 \in R_{prv} \wedge r_1 \succeq r_2) \Rightarrow (owner\_role(r_1) = owner\_role(r_2))$$

- Private roles cannot inherit public roles.

- Public roles can inherit private roles only if both owned by the same tenant.

$$(r_1 \in R_{pub} \wedge r_2 \in R_{prv} \wedge r_1 \succeq r_2) \Rightarrow (owner\_role(r_1) = owner\_role(r_2))$$

- Public roles can inherit public roles from trusted tenants in the circle[1].

$$(r_1, r_2 \in R_{pub} \wedge r_1 \succeq r_2) \Rightarrow (owner\_role(r_1) = owner\_role(r_2) \vee$$
$$owner\_role(r_1) \lhd_\epsilon owner\_role(r_2) \vee owner\_role(r_2) \lhd_\zeta owner\_role(r_1))$$

Essentially private roles inherit private roles in a tenant whereas public roles can inherit private and public roles in a tenant and trusted tenants' public roles.

In our model, *assigned_user_roles* function gives the roles assigned to each user in the circle. The permissions assigned to each role in the model is defined *assigned_permissions* function. A user is assigned to a set of roles and each role is assigned to a set of permissions given by *authorised_user_permissions* function in MT-RBAC$_c$.

6.2.2 Formal MT-RBAC$_c$ Model

We summarize the above in the following formal definition.

Definition 15. *MT-RBAC$_c$ is defined with the following basic component sets and functions.*

- T, U, R, OPS, *and* OBS *(tenants, users, roles, operations, and objects, respectively).*

- $t \in T$, $u \in U$, $r \in R$, $op \in OPS$, *and* $ob \in OBS$.

- R_{pub} *is a set of public roles and* R_{prv} *is a set of private roles where* $R_{pub} \subseteq R$, $R_{prv} \subseteq R$, *and* $R_{pub} \cap R_{prv} = \emptyset$, $R_{pub} \cup R_{prv} = R$

- $PRMS = OPS \times OBS$ *, the set of permissions.*[2]

[1] Trust is defined as a reflexive relation consequently each tenant trusts itself.

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

- $UO \subseteq U \times T$, *a many-to-one user-to-tenant owner relation.*
- $RO \subseteq R \times T$, *a many-to-one role-to-tenant owner relation.*
- $OO \subseteq OBS \times T$, *a many-to-one object-to-tenant owner relation.*
- $owner\_user : (u : U) \rightarrow T$, *the mapping of user* u *into its owner tenant. Formally:* $owner\_user(u) = t$ *where* $(u, t) \in UO$.
- $owner\_role : (r : R) \rightarrow T$, *the mapping of role* r *into its owner tenant. Formally:* $owner\_role(r) = t$ *where* $(r, t) \in RO$.
- $owner\_object : (ob : OBS) \rightarrow T$, *the mapping of object* ob *into its owner tenant. Formally:* $owner\_object(ob) = t$ *where* $(ob, t) \in OO$.
- $CoT \subseteq T$, *is a subset of* T *called Circle-of-Trust. For every two tenants that are member of* CoT $(t_1, t_2 \in CoT)$ *trust relationship is written as* $t_1 \lhd t_2$, *which is symmetric so* $t_1 \lhd t_2$ *iff* $t_2 \lhd t_1$, *reflexive so* $t_1 \lhd t_1$, *and transitive.*
- $HomogeneousCoT_\epsilon$, *for all tenants* t_1 *where* $t_1 \lhd_\epsilon t_2$, *tenant* t_1 *is authorized to assign its users to public roles in* t_2. *Tenant* t_1 *controls* t_1*'s users to* t_2*'s roles assignments.*
- $HomogeneousCoT_\zeta$, *for all tenants* t_1 *where* $t_1 \lhd_\zeta t_2$, *tenant* t_2 *is authorized to assign users from* t_1 *to its public roles. Tenant* t_2 *controls* t_1*'s users to* t_2*'s roles assignments.*
- $UA \subseteq U \times R$, *a many-to-many mapping user-to-role assignment relation requiring that* $(u, r) \in UA \Rightarrow (owner\_user(u) = owner\_role(r) \wedge r \in R_{prv}) \vee ((owner\_user(u) \lhd_\epsilon owner\_role(r) \vee owner\_role(r) \lhd_\zeta owner\_user(u)) \wedge r \in R_{pub})$.
- $PA \subseteq PRMS \times R$, *a many-to-many mapping permission-to-role assignment relation requiring that* $((op, ob), r) \in PA \Rightarrow (owner\_object(ob) = owner\_role(r) \wedge r \in R_{prv})$.
- $RH \subseteq R \times R$ *is a partial order on* R *called hierarchy relation, written as* $\succeq$, *where* $r_1 \succeq r_2$ *requiring that* $(r_1, r_2) \in RH \Rightarrow ((owner\_role(r_1) = owner\_role(r_2)) \wedge \neg(r_1 \in R_{prv} \wedge$

$r_2 \in R_{pub})) \vee ((owner\_role(r_1) \lhd_\epsilon$

$owner\_role(r_2) \vee owner\_role(r_2) \lhd_\zeta owner\_role(r_1)) \wedge (r_1, r_2 \in R_{pub}))$.

- $trusted\_tenants : (t : T) \rightarrow 2^T$, *the mapping of a tenant* t *to a set of trusted tenants in circle-of-trust. Formally:* $trusted\_tenants(t) = \{t' \in T | t \lhd t'\}$

- $authorized\_roles : (t : T) \rightarrow 2^R$, *the mapping of a tenant* t *to a set of authorized roles in circle-of-trust. Formally:* $authorized\_roles(t) = \{r \in R \mid role\_owner(r) = t \vee role\_owner(r) \in trusted\_tenants(t)\}$

- $assigned\_user\_roles : (u : U) \rightarrow 2^R$, *the mapping of user* u *into a set of roles. Formally:* $assigned\_user\_roles(u) = \{r \in R \mid (u, r) \in UA\}$.

- $assigned\_permissions : (r : R) \rightarrow 2^{PRMS}$, *the mapping of role* r *into a set of permissions. Formally:* $assigned\_permissions(r) = \{p \in PRMS \mid (p, r) \in PA\}$.

- $authorized\_user\_permissions : (u : U) \rightarrow 2^{PRMS}$, *the mapping of user* u *into a set of permissions.* $authorized\_user\_permissions(u) =$ $\bigcup_{r \in assigned\_user\_roles(u)} assigned\_permissions(r)$.

Homogeneous Circle Use Case

To better clarify the concept, we present a use-case adopting MT-RBAC$_c$ in Figure 6.3. A group of institutions seeks to collaborate on a cyber security research program where researchers across these universities must have access to dispersed data and computing resources, and be able to collaborate on cross-projects within an association of universities. Universities establish a homogeneous Cyber circle. Each collaborating research institute become a member of Cyber circle which is homogeneous type-ϵ and partners are equally authorized to make user assignment assertions. In this scenario researchers can be assigned to research data, computation resources. In Figure 6.3, UTSA doesn't share its private VMs in the circle by assigning it to private roles only. Assignments are administered by user-owner tenants consistently within the circle. UTA, UTSA, and UTD are

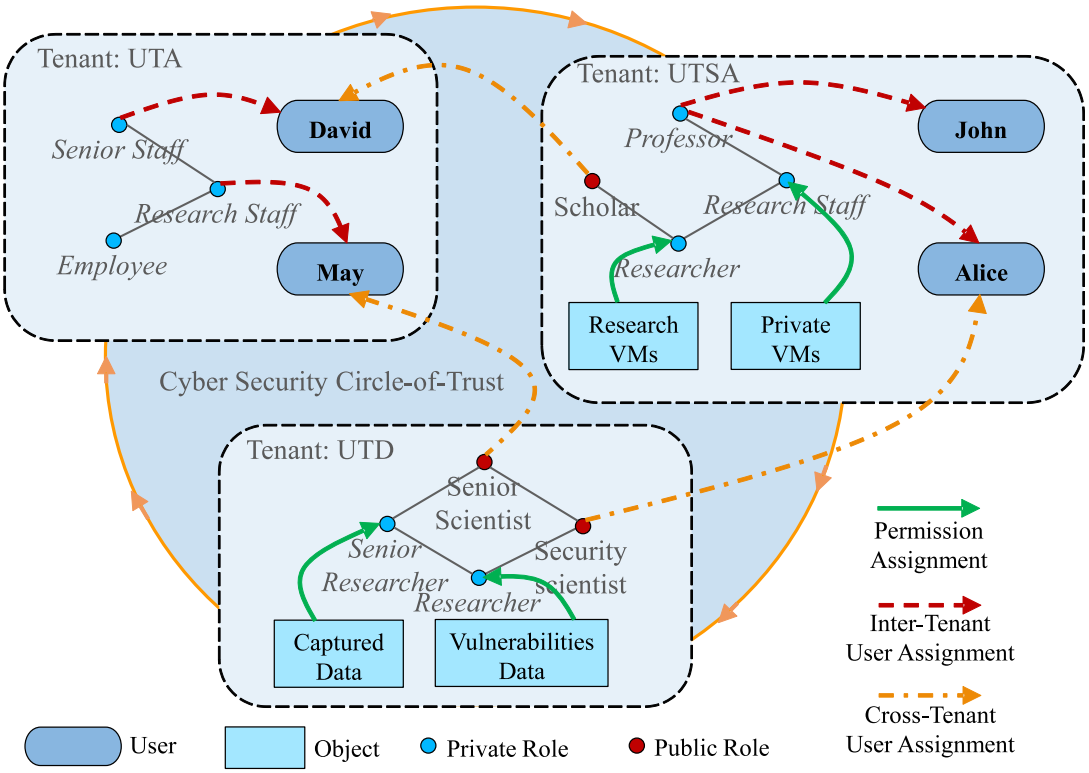


Figure 6.3: Example of Multi-Tenant RBAC$_c$ in Homogeneous circle-of-Trust.

tenants representing respective organizations in a public cloud platform establishing Cyber circle.

6.3 Heterogeneous Role and Attribute-Based Circle-of-Trust

This section, presents a formal multi-tenant role-centric attribute-based access control model designated as MT-RABAC$_c$, providing collaboration in a heterogeneous circle-of-trust. Our model is motivated by previously defined role-centric model [32]. In a heterogeneous circle, entities are not equally authorized to make assertions. Further, each tenant is authorized with respect to its tenant type (or tenant domain). In MT-RABAC$_c$, cross-tenant user assignments are conditional with respect to tenant domain attribute. Attributes are name:value pairs presenting entities' properties in the cloud.

6.3.1 Multi-Tenant Role-Centric Attribute-Based Circle-of-Trust (MT-RABAC$_c$)

MT-RABAC$_c$ adds attributes to enforce cross-tenant assignment separation. Attributes are used to denote tenant types where tenants are authorized to assert cross-tenant user assignments on certain type of tenants.

Figure 6.4 depicts elements in MT-RABAC$_c$ where *tenant attributes* ($TATT$), *user attributes* ($UATT$), and *object attributes* ($OATT$) are added to the tenant, user, and object components 6.2 respectively. *Attribute* is considered as a function which takes tenant, user, or object as input and return a value from its range. For example, an atomic-valued user attribute function such as $employeeType$ returns employee status of a user $john$ where $emplyeeType \in UATT$, $john \in U$ and *emplyeeType(john)* = $full\_time$. Range or scope of an attribute is a finite set of atomic values specifying the valid range of attribute functions. Attribute functions either return a single value or set of values which it refers to as *atomic-valued* and *set-valued* attribute types.

In MT-RABAC$_c$, users and objects are associated with attributes $UATT$ and $OATT$. Each user is assigned a finite set of user attributes such as name, position, salary, etc. Each user attribute is uniquely owned by a tenant, depicted by required meta-attribute $uattOwner$. User attributes are defined as partial function where a user u can be assigned a value for $uatt$ only if

$$user\_owner(u) = uattOwner(uatt)$$

We defined user and object attributes per tenant to eliminate attribute conflict in presence of multi-tenancy. Each object is associated with set of $OATT$ representing properties such as creation time, risk level, VM owners, etc. Each $oatt$ is owned uniquely by a tenant. We defined *oattOwner* as a required atomic-valued meta-attribute mapping $oatt$ to attribute owner tenant respectively. Object attributes are similarly defined as partial function defined only if

$$user\_owner(u) = uattOwner(uatt)$$

User and object attributes can be used by security administrators to further customize access to shared resources, such as a condition where users with management position attribute can only be assigned to management roles in trusted tenants.

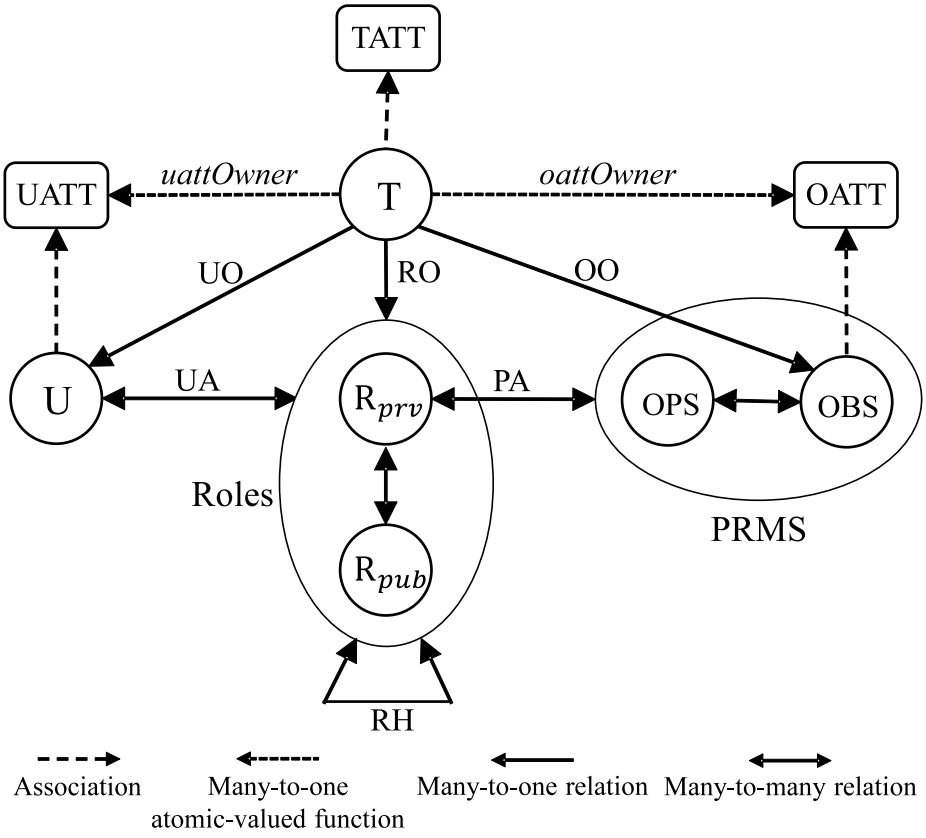


Figure 6.4: Multi-Tenant Role-Centric ABAC Circle-of-Trust.

Tenant attributes are fundamental to MT-RABAC$_c$ to enforce limitations on cross-tenant user assignments. Moreover, we define *domain* (D) as a set of tenant types in the system. Particularly a tenant is related to a domain with an atomic-valued required attribute function *tenantDomain*. It specifies type of a tenant. In order to separate user-assignments, *trustedDomains* is defined as a required set-valued tenant attribute specifying group of tenants which can assert assignments. In this context, user-assignment is modified with respect to trustedDomains attributes. In type-ϵ circle, user-owner tenant can assign its users to roles in tenants which it is a member of trustedDomains set. In typeζ, user assignment is modified to satisfy the condition where role-owner tenant can assign users from tenants in the circle, if it is member of their trustedDomains set. A user is assigned to a role only if

$$(owner\_user(u) = owner\_role(r) \wedge r \in R) \vee$$
$$(owner\_user(u) \lhd_{\epsilon} owner\_role(r) \wedge r \in R_{pub} \wedge$$
$$tenantDomain(owner\_user(u)) \in trustedDomains(owner\_role(r))) \vee$$
$$(owner\_user(u) \lhd_{\zeta} owner\_role(r) \wedge r \in R_{pub} \wedge$$
$$tenantDomain(owner\_role(r)) \in trustedDomains(owner\_user(u)))$$

In a circle-of-trust we allow only one trust type in the circle. We don't allow both type ϵ and ζ at once to a circle due to conflict of interest. Permission-assignment remains unchanged where a permission is assigned to a role only if

$$(owner\_role(r) = owner\_object(o) \wedge r \in R_{prv})$$

Authorized_user_permisisons denotes the set of permissions available to a user where limited user assignments with respect to tenant types reflected.

6.3.2 Formal MT-RABAC$_c$ Model

We formally define MT-RABAC$_c$ as follows.

Definition 16. *Multi-tenant role-centric ABAC$_c$ is defined by the following enhancements and modifications to MT-RBAC$_c$.*

- *T, U, R_{prv}, R_{pub}, OPS, and OBS are defined as in MT-RBAC$_c$.*

- *D represents a finite set of existing domains.*
- *$TATT$, $UATT$, and $OATT$ represent finite set of tenant, user, and object attribute functions respectively.*
- *For each att in $TATT \cup UATT \cup OATT$, $Scope(att)$ represents the attribute's scope, a finite set of atomic values.*
- *$attType : TATT \cup UATT \cup OATT \rightarrow \{set, atomic\}$, specifies attributes as set or atomic valued.*
- *UO, RO, and OO represents user, role, and object owner many-to-one relations as defined in MT-RBAC$_c$.*
- *$owner\_user$, $owner\_role$, and $owner\_object$ are functions mapping users, roles, and objects to owner tenant respectively, defined in MT-RBAC$_c$ model.*
- *Each tenant attribute function maps elements in T to atomic or set values as follows.*

$$\forall tatt \in TATT.tatt : T \rightarrow \begin{cases} Scope(tatt) & if\ attType(tatt) = atomic \\ 2^{Scope(tatt)} & if\ attType(tatt) = set \end{cases}$$

- *Each user attribute function $uatt \in UATT$ is defined as a partial function mapping elements in U to atomic or set values.*

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

$uatt(u : U)$ is defined only if $uattOwner(uatt) = owner\_user(u)$.

- *Each object attribute function $oatt \in OATT$ is defined a partial function mapping elements in O to atomic or set values.*

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

$oatt(o : O)$ is defined only if $oattOwner(oatt) = owner\_object(o)$.

- $MATT = \{uattOwner, oattOwner\}$, *required meta-attribute functions.*
  - $uattOwner : (uatt : UATT) \rightarrow T$, *required user atomic meta attribute function, mapping user attribute* $uatt$ *to attribute owner tenant* t.
  - $oattOwner : (oatt : OATT) \rightarrow T$, *required object atomic meta attribute function, mapping object attribute* $oatt$ *to attribute owner tenant* t.
- $tenantDomain : (t : T) \rightarrow D$, *required tenant atomic attribute function mapping tenant* t *to tenant domain* d. $tenantDomain \in TATT$.
- $trustedDomains : (t : T) \rightarrow 2^D$, *required tenant set attribute function mapping tenant* t *to powerset of trusted domains* D. $trustedDomains \in TATT$.
- $HeterogeneousCoT_\epsilon$, *for all tenants* t_1 *where* $t_1 \lhd_\epsilon t_2$, *if* $tenantDomain(t_1) \in trustedDomain(t_2)$, *then tenant* t_1 *is authorized to assign its users to public roles in* t_2. *Tenant* t_1 *controls* t_1*'s users to* t_2*'s roles assignments.*
- $HeterogeneousCoT_\zeta$, *for all tenants* t_1 *where* $t_1 \lhd_\zeta t_2$, *if* $tenantDomain(t_2) \in trustedDomain(t_1)$, *then tenant* t_2 *is authorized to assign users from* t_1 *to its public roles. Tenant* t_2 *controls* t_1*'s users to* t_2*'s roles assignments.*
- $UA \subseteq U \times R$, *a many-to-many mapping user-to-role assignment relation requiring that* $(u, r) \in UA \Rightarrow (owner\_user(u) = owner\_role(r) \wedge r \in R) \vee (owner\_user(u) \lhd_\epsilon owner\_role(r) \wedge r \in R_{pub} \wedge tenantDomain(owner\_user(u)) \in trustedDomains(owner\_role(r))) \vee (owner\_user(u) \lhd_\zeta owner\_role(r) \wedge r \in R_{pub} \wedge tenantDomain(owner\_role(r)) \in trustedDomains(owner\_user(u)))$.
- $PA \subseteq PRMS \times R$, *a many-to-many mapping permission-to-role assignment relation requiring that* $((op, ob), r) \in PA \Rightarrow (owner\_object(ob) = owner\_role(r) \wedge r \in R_{prv})$.

Chapter 7: CONCLUSION

The following sections summarize contributions of this dissertation and the discuss of some future research directions for future studies.

7.1 Summary

In this research, we introduced a federation framework for Peer-to-Peer and Circle-of-Trust federation models providing authorization federation in multi-tenant cloud IaaS. In Peer-to-Peer federation, a tenant establishes trust to another tenant, however, in Circle-of-Trust a tenant establishes trust with a group of tenants. Defined tenant-trust types enable user-role and attribute assignments across tenants in a cloud IaaS platform.

In Peer-to-Peer federation, we define a multi-cloud MT-RBAC model providing user-role assignments across tenants in distinct clouds and MT-ABAC model enabling attribute assignments to provide access to shared resources across tenants. In multi-cloud MT-RBAC, tenant-trust type-α authorizes trustor tenant to assign trustee's users to its roles and type-β authorizes trustee tenant to assign trustor's users to its roles. Type-γ authorizes trustee tenant to assign its users to roles in trustor and type-δ delegates trustee tenant to assign users to roles in trustor. MT-ABAC considers cross-tenant attribute assignments authorized by type of tenant-trust. Type-α tenant-trust authorizes trustor tenant to assign its attributes to trustee's users. Type-β authorizes a trustee tenant to assign its attributes to trustor's users. Type-γ authorizes trustee tenant to assign trustor tenants' attributes to its users. Tenant-trust in Peer-to-Peer federation considered as unilateral, unidirectional, and non-transitive trust relationship.

In Circle-of-Trust federation, we defined MT-RBAC$_c$ and MT-RABAC$_c$ providing user-role assignments in homogeneous and heterogeneous circles respectively. MT-RBAC$_c$ allows tenants to equally assert cross-tenant user assignments in homogeneous circles where tenants are from uniform type. Further, in MT-RABAC$_c$ attributes are associated with user, object, and tenant components to differentiate user-assignments in heterogeneous circles where tenants are not from uni-

form types. Tenant-trust in the circle is defined between tenant members in the circle as type ϵ and ζ. Type-ϵ tenant-trust authorizes user-owner tenants to assign their users to public-roles in the circle and type-ζ authorizes role-owner tenants to assign users in the circle to their public-roles. Moreover, tenant-trust is considered as multilateral, bidirectional, and transitive trust relations in homogeneous circles and multilateral, bidirectional, and transitive trust relationships in heterogeneous circles.

7.2 Future Work

There are several topics of interest to extend the work presented in this dissertation.

In Multi-tenant multi-cloud federation, extension of current approaches to heterogeneous cloud platforms in addition to policy integration issues in heterogeneous multi-cloud IaaS is a topic of interest to be investigated.

Moreover, multi-tenant ABAC can be explored to cover contextual (risk adaptive) and environmental attributes. Administrative model is another motivating extension of MT-ABAC.

Further, multi-tenant access control models using provenance can become another potential multi-tenant model.

BIBLIOGRAPHY

[1] Amazon AWS. *Amazon*. `http://aws.amazon.com/`.

[2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[3] Michael Armbrust, O Fox, Rean Griffith, Anthony D Joseph, Y Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. M.: Above the clouds: A berkeley view of cloud computing. Technical report, University of California, 2009.

[4] David Bernstein and Deepak Vij. Intercloud security considerations. In *Proceedings of the Second International Conference on Cloud Computing Technology and Science (CloudCom 2010)*, pages 537–544. IEEE, 2010.

[5] Latifa Boursas and Vitalian A Danciu. Dynamic inter-organizational cooperation setup in Circle-of-Trust environments. In *Proceedings of the IEEE Symposium on Network Operations and Management (NOMS 2008)*, pages 113–120. IEEE, 2008.

[6] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[7] Patricia Arias Cabarcos, Florina Almenárez Mendoza, Andrés Marín-López, and Daniel Díaz-Sánchez. Enabling SAML for dynamic identity federation management. In *Wireless and mobile networking*, pages 173–184. Springer, 2009.

[8] 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*, 6:48–55, 2010.

[9] Scott Cantor, Jeff Hodges, John Kemp, and Peter Thompson. Liberty id-ff architecture overview. *Wason, Thomas (Herausgeber): Liberty Alliance Project Version*, 1, 2003.

[10] David W Chadwick. Federated identity management. In *Foundations of Security Analysis and Design V*, pages 96–120. Springer, 2009.

[11] David W Chadwick, Kristy Siu, Craig Lee, Yann Fouillat, and Damien Germonville. Adding Federated Identity Management to OpenStack. *Journal of Grid Computing*, 12(1):3–27, 2014.

[12] Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph Von Praun, and Vivek Sarkar. X10: an object-oriented approach to non-uniform cluster computing. *ACM Sigplan Notices*, 40(10):519–538, 2005.

[13] CometCloud. *CometCloud*. `http://nsfcac.rutgers.edu/CometCloud/node/7`.

[14] Ed Coyne and Timothy R Weil. ABAC and RBAC: Scalable, flexible, and auditable access management. *IT Professional*, 3:14–16, 2013.

[15] Maarten Decat, Jasper Bogaerts, Bert Lagaisse, and Wouter Joosen. Amusa: Middleware for Efficient Access Control Management of Multi-tenant SaaS Applications. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC 2015)*, pages 2141–2148, New York, NY, USA, 2015. ACM.

[16] Maarten Decat, Bert Lagaisse, Dimitri Van Landuyt, Bruno Crispo, and Wouter Joosen. Federated authorization for software-as-a-service applications. In *Proceedings of the Confederated International Conferences: On the move to meaningful internet systems: (OTM 2013 Conferences)*, pages 342–359. Springer, 2013.

[17] 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, 2001.

[18] R.T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000, University of California, Irvine.

[19] Bill Fisher, Norman Brickman, Santos Jha, et al. Attribute Based Access Control Approach, Architecture, and Security Characteristics. *NIST Special Publication*, 1800-3b:52, 2015.

[20] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. (GCE 2008)*, pages 1–10. Ieee, 2008.

[21] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

[22] EITHER EXPRESSLY OR IMPLIEDLY GRANT and A LICENSE TO ANY INTELLECTUAL. *Web Services Federation Language (WS-Federation)*. Citeseer, 2003.

[23] Nikolay Grozev and Rajkumar Buyya. Inter-cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, 44(3):369–390, 2014.

[24] Dick Hardt. The oauth 2.0 authorization framework, 2012.

[25] Micheal Hogan, Fang Liu, Annie Sokol, and Jin Tong. *The NIST definition of cloud computing*, 2011.

[26] HPE Helion Eucalyptus. *Eucalyptus*. `http://www8.hp.com/us/en/cloud/helion-eucalyptus-overview.html`.

[27] Vincent C Hu, David Ferraiolo, et al. Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication*, 800:162, 2014.

[28] Vincent C Hu, D Richard Kuhn, and David F Ferraiolo. Attribute-based access control. *Computer*, 2:85–88, 2015.

[29] Jingwei Huang, David M. Nicol, Rakesh Bobba, and Jun Ho Huh. A Framework Integrating Attribute-based Policies into Role-based Access Control. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies (SACMAT 2012)*, SACMAT '12, pages 187–196, New York, NY, USA, 2012. ACM.

[30] John Hughes and Eve Maler. Security assertion markup language (saml) v2. 0 technical overview. *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08*, pages 29–38, 2005.

[31] Xin Jin, Ram Krishnan, and Ravi Sandhu. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In Nora Cuppens-Boulahia, Frédéric Cuppens, and Joaquin Garcia-Alfaro, editors, *Proceedings of the 26th Annual Conference on Data and Applications Security and Privacy XXVI (IFIP WG 2012)*, pages 41–55, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[32] Xin Jin, Ravi Sandhu, and Ram Krishnan. RABAC: Role-Centric Attribute-Based Access Control. In Igor Kotenko and Victor Skormin, editors, *Proceedings of the Sixth International Conference on Mathematical Methods, Models and Architectures for Computer Network Security (MMM-ACNS 2012)*, pages 84–96, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[33] D. R. Kuhn, E. J. Coyne, and T. R. Weil. Adding Attributes to Role-Based Access Control. *Computer*, 43(6):79–81, June 2010.

[34] Anil Kurmus, Moitrayee Gupta, Roman Pletka, Christian Cachin, and Robert Haas. A comparison of secure multi-tenancy architectures for filesystem storage clouds. In *Proceedings of the 12th International Middleware Conference (Middleware 2011)*, pages 460–479. International Federation for Information Processing, 2011.

[35] Tobias Kurze, Markus Klems, David Bermbach, Alexander Lenk, Stefan Tai, and Marcel Kunze. Cloud federation. *CLOUD COMPUTING*, 2011:32–38, 2011.

[36] Uwe Kylau, Ivonne Thomas, Michael Menzel, and Christoph Meinel. Trust requirements in identity federation topologies. In *Proceedings of the International Conference on Advanced Information Networking and Applications, 2009. (AINA 2009)*, pages 137–145. IEEE, 2009.

[37] Marcos AP Leandro, Tiago J Nascimento, Daniel R dos Santos, Carla M Westphall, and Carlos B Westphall. Multi-tenancy authorization system with federated identity for cloud-based environments using shibboleth. In *Proceedings of the Eleventh International Conference on Networks*, pages 88–93, 2012.

[38] 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.

[39] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing-The business perspective. *Decision Support Systems*, 51(1):176–189, 2011.

[40] Peter Mell and Tim Grance. *The NIST definition of cloud computing*, 2011.

[41] Microsoft Azure. *Microsoft*. http://azure.microsoft.com/.

[42] RL Morgan, Scott Cantor, Steven Carmody, Walter Hoehn, and Ken Klingenstein. Federated security: The shibboleth approach. *Educause Quarterly*, 27(4):12–17, 2004.

[43] S. K. Nair, S. Porwal, T. Dimitrakos, A. J. Ferrer, J. Tordsson, T. Sharif, C. Sheridan, M. Rajarajan, and A. U. Khan. Towards secure cloud bursting, brokerage and aggregation. In *Proceedings of the 8th European Conference on Web Services (ECOWS 2010)*, pages 189–196. IEEE, Dec 2010.

[44] Canh Ngo, Yuri Demchenko, and Cees de Laat. Multi-tenant attribute-based access control for cloud infrastructure services. *Journal of Information Security and Applications*, 2015.

[45] Virtual Organization Summary Open Science Grid. *Virtual Organization*. http://myosg.grid.iu.edu/vosummary?all_vos=on&active=on&active_value=1&datasource=summary.

[46] OpenID. *OpenID*. http://openid.net/.

[47] OpenNebula. *OpenNebula*. http://www.openebula.org/.

[48] OpenStack. *OpenStack*. http://www.openstack.org/.

[49] Openstack-Icehouse. *Icehouse*. http://www.openstack.org/software/icehouse/.

[50] Dana Petcu. Multi-Cloud: Expectations and Current Approaches. In *Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds (Multi-Cloud 2013)*, pages 1–6, New York, NY, USA, 2013. ACM.

[51] Rajiv Ranjan and Rajkumar Buyya. Decentralized overlay for federation of enterprise clouds. *Handbook of Research on Scalable Computing Technologies*, 191, 2009.

[52] Rajiv Ranjan, Aaron Harwood, Rajkumar Buyya, et al. Grid federation: An economy based, scalable distributed resource management system for large-scale resource coupling. *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia*, 2004.

[53] Ravi Sandhu. The authorization leap from rights to attributes: Maturation or chaos? In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies (SACMAT 2012)*, pages 69–70, New York, NY, USA, 2012. ACM.

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

[55] Mukesh Singhal, Santosh Chandrasekhar, Tingjian Ge, Ravi S Sandhu, Ram Krishnan, Gail-Joon Ahn, and Elisa Bertino. Collaboration in Multicloud Computing Environments: Framework and Security Issues. *IEEE Computer*, 46(2):76–84, 2013.

[56] Hassan Takabi, James BD Joshi, and Gail-Joon Ahn. Security and privacy challenges in cloud computing environments. *IEEE Security & Privacy*, 6:24–31, 2010.

[57] Bo Tang. *Multi-Tenant Access Control for Cloud Services*. PhD thesis, University of Texas at San Antonio, 2014.

[58] Bo Tang, Qi Li, and Ravi Sandhu. A multi-tenant RBAC model for collaborative cloud services. In *Proceedings of the Eleventh Annual International Conference on Privacy, Security and Trust (PST 2013)*, pages 229–238. IEEE, 2013.

[59] Bo Tang and Ravi Sandhu. Cross-tenant trust models in cloud computing. In *Proceedings of the 14th IEEE Information Reuse and Integration (IRI 2013)*, pages 129–136. IEEE, 2013.

[60] Bo Tang and Ravi Sandhu. Extending OpenStack Access Control with Domain Trust. In *Proceedings of the eighth International Conference on Network and System Security (NSS 2014)*, page 15. springer, 2014.

[61] Bo Tang, Ravi Sandhu, and Qi Li. Multi-tenancy authorization models for collaborative cloud services. In *Proceedings of the International Conference on Collaboration Technologies and Systems (CTS 2013)*, pages 132–138. IEEE, 2013.

[62] William Tolone, Gail-Joon Ahn, Tanusree Pai, and Seng-Phil Hong. Access Control in Collaborative Systems. *ACM Computing Surveys (CSUR)*, 37(1):29–41, March 2005.

[63] Adel Nadjaran Toosi, Rodrigo N. Calheiros, and Rajkumar Buyya. Interconnected Cloud Computing Environments: Challenges, Taxonomy, and Survey. *ACM Computing Surveys (CSUR)*, 47(1):7:1–7:47, May 2014.

[64] 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, 2006 (CollaborateCom 2006)*, pages 1–9. IEEE, 2006.

VITA

Navid Pustchi was born in Maryland, United States. He received his B.S. in 2007. He subsequently entered Master's program at University of Texas at San Antonio in fall 2009 and started the doctoral program in 2012. At the university, he joined the Institute for Cyber Security and has been working closely under the supervision of Dr. Ravi Sandhu. His research interests include security and privacy in cyber space. In particular, his focus is on developing federation mechanisms in role-based and attribute-based access control and cloud Infrastructure-as-a-Service systems.