Securing Loosely-Coupled Collaboration in Cloud Environment through Dynamic Detection and Removal of Access Conflicts

Nirnay Ghosh, *Student Member, IEEE*, Debangshu Chatterjee,
Soumya K Ghosh, *Member, IEEE*, and Sajal K Das, *Senior Member, IEEE*

Abstract—Online collaboration service has become a popular offering of present day Software-as-a-Service (SaaS) clouds. It facilitates sharing of information among multiple participating domains and accessing them from remote locations. Owing to loosely-coupled nature of such collaborations, access request from a remote user is made in the form of a set of permissions. The cloud vendor maps the requested permissions into appropriate local roles in order to allow resource access. However, coexistence of such multiple simultaneous role activation requests may introduce conflicts which violate the principle of security. In this paper, we propose a distributed secure collaboration framework which enables collaborating domains to detect and remove these conflicts. Two features of our framework are: (i) it requires only local information, and (ii) it detects and removes conflicts on-the-fly. Formal proofs have been provided to establish the correctness of our approach. Experimental results and qualitative comparison with related work demonstrate the efficacy of our approach in terms of response time, thus addressing the scalability requirement of cloud services.

Index Terms—Cloud, collaboration service, loosely-coupled, access conflict, role hierarchy, separation of duty

1 Introduction

Cloud computing is an emerging paradigm which enables customers to use resources as services. Three prominent delivery models supported by cloud computing are: (i) *Infrastructure-as-a-Service (IaaS)*, (ii) *Platform-as-a-Service (PaaS)*, and (iii) *SaaS*. One of the popular offerings of the SaaS cloud is *online collaboration service* [1], [2]. Online collaboration tools enable various organizations to share and access information in a faster and more seamless manner. A few examples of such service include document-centric collaboration, project management, blogs, micro blogging, wikipages, feeds from social networks, file sharing and synchronization, and so on. A common concern in a collaborative environment (e.g., cloud) is that the integrity of data, shared across multiple users, may be compromised [3]. Moreover, choosing an ideal vendor to provide secure and guaranteed collaboration service is also non-trivial [4]. Nevertheless, collaboration has become not only valuable but also essential as it allows the organizations to easily connect with partners, customers, and employees from remote locations with less communication latency. A recent Forrester[1] survey found that more than 56 percent of software decision-makers are using or expected to use SaaS offerings to replace or complement their existing collaboration technology. Cloud-based collaboration provides storage spaces to multiple organizations where they can share resources, and access them through policies formed by customizing the available APIs. Some noteworthy collaboration service providers[2] are: (i) *Acrobat.com*, (ii) *Box.net*, (iii) *CubeTree*, (iv) *HyperOffice*, (v) *Google Apps*, (vi) *MS Office Live*, (vii) *Zoho*, and so on. Collaboration among multiple domains is either tightly or loosely-coupled [5], [6]:

- *Tightly-coupled collaboration*. In this collaboration, there exists a master domain which mediates accesses to individual local domains through a global policy which is formed by integrating policies from participating domains. In such collaboration, interoperation needs are static and predefined.
- *Loosely-coupled collaboration*. In this collaboration, independent systems dynamically come together to share information for a period of time. No global policy is maintained as interoperation requests are "on-demand" to facilitate dynamic data sharing.

In a cloud environment, both tight and loose coupling may take place depending on the nature of collaboration. For example, if different departments of an organization

- *N. Ghosh, D. Chatterjee, and S.K. Ghosh are with the School of Information Technology, Indian Institute of Technology, Kharagpur 721302, India. E-mail: {nirnay.ghosh, deba2510}@gmail.com, skg@iitkgp.ac.in.*
- *S. K. Das is with the Department of Computer Science, Missouri University of Science and Technology, Rolla, MO 65409, USA. E-mail: sdas@mstedu.*

Manuscript received 11 Apr. 2014; revised 31 July 2014; accepted 3 Sept. 2014. Date of publication 3 Oct. 2014; date of current version 7 Sept. 2016. Recommended for acceptance by J. H. Abawajy.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2014.2361527

1. Available online at: http://www.informationweek.com/cloud-computing/software/cloud-collaboration-tools-big-hopes-big/240143787
2. Available online at: http://cloudtaxonomy.opencrowd.com/taxonomy/software-as-a-service/collaboration/

collaborate using cloud services, it is an instance of tightly-coupled collaboration. However, if autonomous domains collaborate "on-demand" for a limited period of time, it is an example of loosely-coupled collaboration.

For both collaborations, if multiple collaboration requests are generated within the same period of time or during a particular user session, it may introduce two types of conflicts [6], [7]:

1) *Cyclic inheritance conflict*. Cross-domain hierarchical relationship may introduce a cycle which enables a subject lower (or junior) in the hierarchy, to acquire permissions of the subjects higher (or senior) in the hierarchy.
2) *Violation of separation of duty (SoD) constraints*. SoD prevent two or more subjects from accessing an object that lies within their conflict of interests or disallow a subject from accessing conflicting objects or permissions. Violations of such constraints may occur because of the interplay of various policy constraints across domains.

Owing to such conflicts, a user which has currently activated a role in the local domain, either inherits permissions of a senior role in the hierarchy, or gets permissions of a conflicting role, respectively [6], [7]. This leads to violation of the principle of security [8] (explained in Section 3). Removal of these conflicts have been reported in the literature, where either global policies have been designed or trust management approaches have been adopted to govern the level of access. Both of these approaches target conflict mediation in tightly-coupled collaboration and have certain limitations with respect to loosely-coupled one [6] (refer to Section 2.4). These limitations of access conflict mediation in loosely-coupled collaboration motivates our work.

The focus of this paper is on *access conflict detection and removal in loosely-coupled collaboration in cloud environments*. To the best of our knowledge, proactive detection and removal of these conflicts in clouds have not been addressed in the existing literature. The contributions of this paper are summarized as follows:

- Propose a distributed collaboration framework to secure interoperation in cloud environment.
- Formally define the concepts of "safe" and "unsafe" cycles that are formed while allowing multiple interoperation requests simultaneously.
- Propose a novel technique to dynamically detect and remove "unsafe" cycles using *local information*.
- Present theoretical proofs for correctness of cyclic inheritance and SoD violation detection mechanisms.
- Analyze the performances of our approach under two situations: (i) variable number of participating domains, and (ii) variable sizes of role hierarchy in domains, with a goal to address the scalability requirement of any cloud-based service.

The rest of the paper is organized as follows: Section 2 provides a background of secure multi-domain collaboration and reviews relevant literature in the related area. Section 3 outlines a motivating example to comprehend the underlying challenge and then draw objectives of this work. Dynamic detection and removal of conflicts are presented in Sections 4 and 5, respectively. Analysis of simulation results is described in Section 6. Finally, Section 7 draws the conclusion with direction of future research.

2 Background and Literature Review

In this section, we present an introduction to security issues in multi-domain collaboration and give a comprehensive review of related work from literature which focus on detection and removal of access conflicts.

Ensuring security in multi-centric collaborative system is non-trivial and is governed by the following principles [8]:

- *Principle of autonomy*. If an access is permitted within an individual system, it must also be permitted under secure interoperation.
- *Principle of security*. If an access is not permitted within an individual system, it must not also be permitted under secure interoperation.

Traditional access control models, such as *Discretionary Access Control (DAC)* [9], *Mandatory Access Control (MAC)* [10], [11], and *Role-based Access Control (RBAC)* [12], [13], have been designed to ensure the above mentioned principles within a single system (or domain). However, they have limitations as far as implementing security in multi-domain interoperation is concerned.

For secure multi-system collaboration, a provider domain has to ensure the following requirements simultaneously:

- *Security*. To guarantee the principle of least privilege [14] which states that, *a user should not be given any more access to resources than is required to process the request*.
- *Fairness*. Ensure availability of the requested set of permissions for honoring the essence of collaboration [15].

To address these issues, we present below several approaches from the literature and also attempt to point out their limitations with respect to loosely-coupled collaboration in a cloud environment.

2.1 Global Policy Based Approaches

In such approaches, policies from individual domains are mapped into a centralized global policy, based on which all interoperation requests are authorized. Examples include computational complexity analysis [8] based on the Access Control Matrix (ACM) model, policy algebra [16], Multi-Level Security (MLS) model [17], and secure interoperation framework [7] based on the RBAC model. Global policy must ensure both principles of autonomy and security.

The use of global policy in multi-domain environments (where RBAC is employed) could introduce two types of violations in principle of security [7]: (i) cyclic inheritance conflict and (ii) violation of SoD constraints (defined in Section 1). Cyclic inheritance conflict occurs when there exists cycles among authorized interoperations and local hierarchical relations. Similarly, violation of a predefined SoD constraint occurs if any user activates conflicting role in the same or concurrent sessions. Both these conflicts can be represented by *cycles* [6], as shown in Figure A.1 of the *supplemental material*, which can be found on the Computer

Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCC.2014.2361527.

In [7], the authors propose a policy integration framework for merging heterogeneous RBAC policies of multiple domains into a global access control policy. An integer programming (IP)-based approach is proposed as the optimal resolution for static constraints. This work has been extended in [18] to resolve conflicts generated from event-driven RBAC policies, viz. role-specific and user-specific SoDs. In [15], a distributed secure interoperability protocol is proposed that ensures secure interoperation of multiple collaborating domains without compromising their security. In [19], the authors present the design and implementation of an administration model that aims at enabling administration of RBAC policies with constraints and coordinates conflict resolution in a multi-domain environment.

2.2 Trust Management in Multi-Domain Environments

Trust management approaches [20], [21], [22], [23] make authorization decisions on dynamic interoperations involving domains previously unknown to each other. Authorization decisions are typically made based on the properties of the entities, and such properties are encoded in *credentials*. In a trust management system, each individual domain specifies several rules defining which credentials are needed to access its resources. External users requesting the resources need to prove that they have the required credentials for an access right over the resources as defined in the policy of the providing domain [24]. Whether a user should be given a certain credential could in turn be defined by the policy of the domain issuing that credential, which may further require another set of credentials to be validated.

2.3 Other Approaches

Several techniques have been proposed for resolution of inheritance conflict in the standard RBAC hierarchy, which may be summarized as follows:

- *Adding a new interface role* [25]. An interface role is junior to the actual requested role and inherits a part of the permissions.
- *Resolution by priority* [7], [26], [27]. Each authorization/prohibition is assigned a priority and lower priority authorization/prohibitions are overridden in case of a conflict. Priorities can be either explicitly assigned to the individual authorization/prohibition or can be derived based on the administrative scope of the grantor [28].
- *Resolution by restriction* [27], [29]. Prevents conflicting access at the expense of restricted accessibility. This restriction may be enforced through: (i) relaxation of autonomy, (ii) relaxation of local privileges or constraints, and (iii) reduction of degree of autonomy.

In [30], [31], the authors identified that the standard RBAC hierarchy lacks flexibility to enforce SoD constraints for preventing a user from acquiring two conflicting roles simultaneously. The *ER-RBAC96 model* [31] distinguishes between permission inheritance and role activation semantics and introduced the concept of hybrid RBAC hierarchy. In [32], three distinct hierarchical relations have been defined: *(i) I-hierarchy (inheritance only), (ii) A-hierarchy (activation-only), and (iii) IA-hierarchy (general-hierarchy)*. It has been shown that such a fine-grained hierarchy can allow specification of a wide range of security requirements, including the specification of Dynamic Separation of Duty (DSoD), and user-centric as well as permission-centric cardinality constraints on roles [32].

In the hybrid RBAC hierarchy [31], some approaches have been reported to resolve cyclic inheritance conflict. These are as follows:

- *Converting an I-relation to A-relation in local role hierarchy* [33]. In an I-hierarchy, a senior role inherits all permissions of a junior role, without activating the later. However, in an A-hierarchy, the senior role can activate a junior role, but cannot inherit the latter's permissions. This will prevent a user who has activated a junior role from inheriting permissions belonging to a senior role in the local domain.
- *Use of access role for cross-domain collaborations* [6]. An access role is a newly created role in the resource providing domain which is A-junior to the requesting role from a remote domain, and I-senior to roles in the local domain. For accessing permissions of a role, an A-relation must precede an I- relation in the role hierarchy.

2.4 Limitations of Existing Approaches

Global policy-based approaches are acceptable in tightly-coupled or federated collaboration, where interoperation requests are predefined, and there exists a master domain which mediates accesses to individual local domains. On contrary, in loosely-coupled collaboration, independent domains dynamically interoperate and reveal only limited information about their services and policies, relevant to collaboration [34]. Consequently, a requester domain is not aware of the policy structure (or role hierarchy) of the provider domain. Under this situation, from RBAC perspective, it is convenient to send interoperation requests in the form of a *set of permissions* [6]. The issue of semantic mismatch, in terms of representation of objects and access modes, is not relevant in cloud context. This is because, collaborating domains have to share their resources and grant access to them by explicitly using the available APIs. Sending collaboration request in the form of set of permissions has been found to prevail in *Google Cloud Storage*[3], where a requester sends access request by generating hashed codes, representing a set of resources and operations. To initiate collaboration, the requested permissions are mapped into roles for activation.

For cloud-based collaboration, the participating domains have to strictly use vendor provided APIs to model access policies. Presently, cloud vendors use token-based authentication (e.g., OAuth 2.0) scheme, eliminating the need for sensitive credential information. Such token uses scopes of access modes (read, write, execute, etc.) to determine the privilege level on the requested object. Generally, a resource provider issues access tokens to the requesting applications,

3. Available online at: https://developers.google.com/storage/docs/collaboration

thus specifying the allowed level of access. At the time of request, the application attaches this access token with the hashed code of object. If the request and the token are found to be valid, access is granted, until the token is timed out. As such authorization is not based on the properties of requesting entity, trust is not generated before granting access to any resource. Therefore, the use of trust management systems is not realistic in cloud scenario.

Limitations with other approaches (discussed in Section 2.3) based on both standard and hybrid RBAC hierarchies are as follows:

- *Adding a new interface role*. A major challenge with this interface role is deciding its permission set.
- *Resolution by priority*. This approach is suitable for systems that are managed by multiple administrators which may specify contradictory rules for accessing a particular request.
- *Resolution by restriction*. The restricted access semantics in this approach may significantly reduce accessibility and lead to a deadlock.

To summarize, dynamic detection of multi-domain access conflict is essential for secure interoperation. In the proposed work, our conflict resolution scheme is aimed at providing secure and fair collaboration among collaborating domains in the cloud environment. For fair collaboration, the objective is to make the requested permissions from a remote domain available. If any request is found to violate local domain's policy constraints, rather than outright deny, it is made available in a controlled manner.

Some works on securing cloud-based collaboration are available in the literature. A summary of those approaches has been presented in Table 1. Stemmed from absence of relevant experimental studies in this area of cloud computing security, this summary can be considered as a qualitative comparison of the proposed method with the reported work.

It is evident from Table 1 that most of the works have not presented experimental results of their proposed security approaches. Some results are given in [38], however, their motivation is to verify the policies, rather than dynamic detection and removal of conflicts. In this work, we evaluated the performance of the framework in terms of response time to establish its applicability as a cloud-based service.

3 A Motivating Example

Consider a cloud, say Google Cloud Storage, where customers store data and run applications in the form of objects and buckets. Google provides APIs for default access control lists (ACLs) to manage accesses to objects and buckets. However, the owners of these resources customize the available APIs according to their security policies to enforce access control mechanism. Each ACL entry consists of the following:

- *Scope*. Refers to a user or a group of users to whom access is granted and is used for authentication purpose.
- *Permission*. Access right that can be performed against an object or bucket.

Scope can be defined by the following authentication entities: (i) Google storage ID, (ii) Google account/group email address, (iii) Google App domain, (iv) special identifier for all Google account holders, and (v) special identifier for all users. Permissions or access modes supported by the Google Cloud Storage are: $FULL\_CONTROL > WRITE > READ$, implying dominance relation among various modes. Any bucket requested by the user gets created under a specific project. Each project is identified by an unique project ID.

Google provides various types of default ACLs to manage the projects. For our scenario, we use *project-private ACL*, by which permissions are assigned to the project team based on their roles. Three roles have been identified by the Google Cloud Storage:

- *Project viewer*. $READ$ access to bucket in a project.
- *Project editor*. $WRITE$ (may be $FULL\_CONTROL$ on some occasions) access to bucket in a project.
- *Project owner*. $FULL\_CONTROL$ access to bucket + administrative tasks (e.g., adding/removing project members, changing billing information, etc.)

When a user/application request accesses an object or bucket, the Google Cloud Storage system reads the ACL on the object or bucket and determines whether to allow or reject the access request. If the ACL grants permission for the requested operation, the Google authorization server generates an *access token* which initiates collaborative activities. If the ACL does not grant permission for the requested operation, the request fails and a 403 Forbidden Error (Access Denied) is returned.

In such an operational scenario, let three geospatial domains $D1$, $D2$, and $D3$ create three projects $P1$, $P2$, and $P3$ respectively (refer Fig. 1). The projects contain buckets and objects which the users of respective domains can access. We consider a simple scenario where the projects contain single bucket, say, $B1$, $B2$, and $B3$, respectively. Let domain $D1$ manages information relevant to road networks in a particular region. Let $D2$ manages health related information, viz. the number of hospitals, specialties, capacity, and so on. Finally, let $D3$ maintains demographic information. Participating domains are independent and implement access policies according to their need. The role hierarchies of collaborating domains are given in Table 2.

Normally, there exist no specific common tasks that require interoperations among these geospatial domains. They are able to function on their own to carry out daily operations without collaborating with each other. Collaboration needs are "on-demand" and one such collaborative scenario may occur at the time of handling a mission-critical application as follows: *suppose a particular region is affected by earthquake, and the local government decided to send a medical team to the site to rescue affected people and move them to proper hospitals for medication. Before sending the team, some prior decisions are to be made such as, which route to follow to reach the affected area in shortest time, how many people are affected, location and capacity of the hospitals where the victims can be admitted, and so on.*

As evident from the above query, all information is not readily available with single geospatial organization, implying the need for dynamic collaboration among domains. This is an example of loosely-coupled collaboration, where

TABLE 1
Comparative Study with Related Work

Reference	Objective	Approach	Results
Almutairi *et al.* 2012 [34]	Securing inter-cloud collaboration which may take either in same layers (IaaS-IaaS, PaaS-PaaS, etc.) or in cross layers (IaaS-PaaS, PaaS-SaaS, etc.).	Distributed access control module (ACM) and virtual resource manager (VRM) have been proposed which are to be implemented in all the layers of cloud. ACM makes authorization decisions based on credentials and context, while VRM provides and deploys virtual resources.	No experiment results given.
Singhal *et al.* 2013 [35]	Ensure security for dynamic collaborations and resource sharing among multiple clouds which do not have any pre-established collaboration agreement or standardized interfaces.	The paper proposed a proxy-based multi-cloud computing framework. A proxy is an edge-node-hosted software instance that a client or a CSP can delegate to carry out operations on its behalf. Proxies can act as mediators for collaboration among services on different clouds.	No experiments have been conducted.
Almorsy *et al.* 2011 [36]	Enforce NIST-FISMA security certificate to make collaboration between cloud provider and consumer secure and trusted.	A cloud security management framework has been implemented which considers the standards specified in NIST-FISMA certificate. This framework consists of three main layers: (i) management, (ii) feedback, and (iii) enforcement.	A proof-of-concept of the framework has been developed using *.NET*. Authors have considered a case study to demonstrate applicability; however, no performance evaluation (based on response time) has been done.
Takabi *et al.* 2012 [37]	Enable users to manage access policies for controlling the access to heterogeneous resources deployed over multiple cloud environments.	The authors propose a framework for policy management service provider (PMSP) that enables users to define, edit and manage accesses. Important steps of operation are: (i) registration to PMSP, (ii) resource discovery, (iii) access policy specification, and (iv) translating and exporting policies to cloud providers.	No experiment results given.
Gouglidis *et al.* 2014 [38]	Verification of multi-domain cloud policies relevant for collaboration.	The authors proposed a model checking technique based on RBAC reasoning. It acts as a management service/tool for verification of access policies from multiple domains. Based on these policies, domains interoperate in a cloud environment. The verification process combines all the policies from different domains and carries out compliance checking. This approach deals with detection of access conflict, but does not handle their removal.	Simulation of collaborating domains and roles done and experiment results showing the performance of the verification process (both normal and optimized) are given. However, performance evaluations with respect to domains or roles is not done.
Calero *et al.* 2010 [39]	Authorizing collaborations among services hosted in different clouds	A third-party authorization system has been developed which functions at two layers: (i) authorization server - consisting of policy decision point (PDP) engine, trust manager, and knowledge base (ii) authorization client API - with GUI and policy enforcement point (PEP).	Experimental results showing the performances of three methods (*hasAuth*, *searchGrant*, *addGrant*) while subjected to multiple simultaneous requesters are given. The results show no impact on searching, but linear deviation for the other two.
Proposed work	Dynamic detection and removal of access conflicts for securing collaboration among multiple domains in cloud environment	A distributed secure collaboration framework is proposed. Before allowing a remote user to activate local roles, this framework locally checks each collaboration request to determine if there is any possibility of an anomaly in future. On detection of a conflict, the framework recommends actions so that the collaboration does not give any unauthorized access.	Experiments have been conducted to address the scalability requirement of cloud-based services. The numbers of participating domains and the roles in each domain have been varied. The framework is observed to detect and remove the conflicts in $O(n^4)$ time.

TABLE 2
Role Hierarchies of Participating Domains

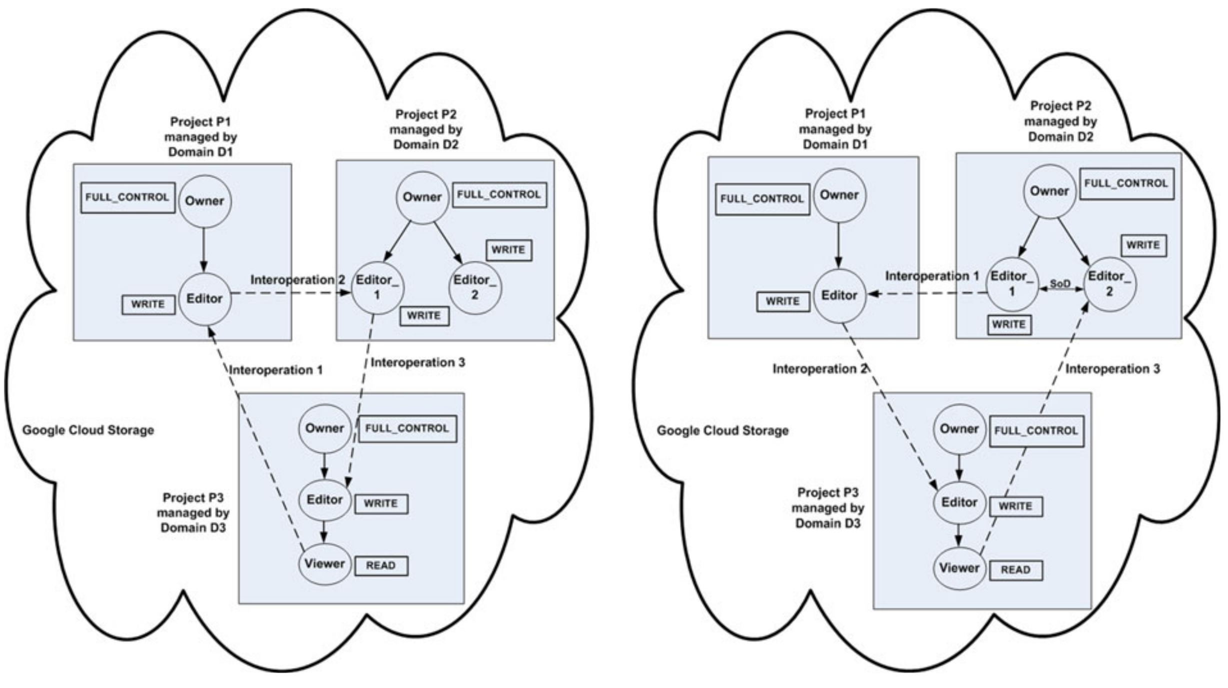
Domain	Project	Roles	Permissions	Parent(s)
$D1$	$P1$	Owner	$FULL\_CONTROL$	–
		Editor	$WRITE$	Owner
$D2$	$P2$	Owner	$FULL\_CONTROL$	–
		Editor_1	$WRITE$	Owner
		Editor_2	$WRITE$	Owner
$D3$	$P3$	Owner	$FULL\_CONTROL$	–
		Editor	$WRITE$	Owner
		Viewer	$READ$	Editor

interoperation needs are dynamic and cannot be predefined. To share information and resources, each owner domain sets up project-private ACL for its project, which regulates the access to its constituent bucket and objects. The project ACL policies are depicted in Table 3. Now, in order to handle the dynamic collaboration request of "sending rescue teams to the earthquake site", let $Viewer_{D3}$ needs to download some objects from bucket $B1$. If such collaboration is authorized, it activates $Editor_{D1}$, which is interoperation 1 in Fig. 1a. If $Editor_{D1}$ needs to obtain some analytical information from $D2$, it has to activate $Editor\_1_{D2}$ depicted by interoperation 2 in Fig. 1a. Let us assume at the same time instant, $Editor\_1_{D2}$ has to download some objects from bucket $B3$, to get information related to population density in the affected region. Hence, to facilitate this interoperation, it activates $Editor_{D3}$, as shown by interoperation 3 in Fig. 1a. At the present time instant, if all three interoperations are authorized, a cycle is introduced. Due to this cycle, a user who has activated *Viewer* role in domain $D3$ also gains the privilege of senior role *Editor*, in the same domain. This is a violation to the principle of security, as such access was prevented in $D3$, prior to interoperation.

Suppose, at some other instant, there exists a role-specific SoD constraint between $Editor\_1_{D2}$ and $Editor\_2_{D2}$ roles in $D2$. Let these roles have $WRITE$ accesses on conflicting objects contained in the bucket $B2$. Such SoD constraint prevents activation of concerned roles by the same user in the same session or in concurrent sessions. Fig. 1b shows three simultaneous interoperation requests made in this scenario. The SoD constraint between two roles in domain $D2$ has been identified by means of a doubled-headed arrow. It is evident from Fig. 1b, a user, who activated $Editor\_1_{D2}$ role, obtains permissions related to $Editor\_2_{D2}$ through successive cross-domain interoperations 1, 2, and 3, respectively. This violates the predefined SoD (separation of duty) constraint. Unlike tightly-coupled collaboration, there is no static global policy in a loosely-coupled environment, which attempts to resolve the conflicts by removing one or more interoperation links. Moreover, as the participating domains are autonomous, there exists no centralized mediator which detects and removes such conflicts. This challenge illustrates that there is a need for proper mechanism to detect and remove cyclic inheritance conflict and violation of SoD constraints in a distributed environment like cloud.

4 Dynamic Detection of Conflicts

In a multi-domain environment, to process an interoperation/collaboration request, one or more local roles have to be activated (after authentication of the requester). We model interoperation request as a sequence of roles (belonging to different domains), that originates from a requester domain and terminates at a provider domain. A cycle is formed when the role sequence has the same source and destination role. Such cycles may introduce violation in the principle of security (as discussed in Sections 2.1). However, all role sequences which form cycles do not necessarily



(a) Cyclic Inheritance Conflict (b) Violation of SoD Constraint

Fig. 1. Motivating example.

TABLE 3
Project ACL Policies

Project	Requester	Scope-type	Email/ID	Permission	Resource
$P1$	$D3$	*GroupByEmail*	*some_id@googlegroups.com*	*WRITE*	Bucket: $B1$
	$D2$	*GroupByID*	*some_id*	*WRITE*	Objects: $O_1, \dots O_k \in B1$
$P2$	$D1$	*GroupByID*	*some_id*	*WRITE*	Bucket: $B2$
	$D3$	*GroupByDomain*	*some_domain_id*	*READ*	Objects: $O_1, \dots O_m \in B2$
$P3$	$D2$	*GroupByDomain*	*some_domain_id*	*WRITE*	Bucket: $B3$
	$D1$	*GroupByEmail*	*some_id@googlegroups.com*	*READ*	Bucket: $B3$

cause security violations. We term them as *safe role sequence*. Conversely, if the cycle violates the principle of security, we refer to them as *unsafe role sequence*. In this section, we present our approach to dynamically detect the conflicts produced by unsafe role sequences.

4.1 Preliminaries

We propose a distributed secure collaboration framework for cloud environment (refer to Fig. 2). The framework is generic and is expected to work for collaboration service offered by other cloud vendors. It will be provided as a service so that collaborating domains, before granting access against an interoperation request, checks the violation of the local policies, if any. Two important assumptions adopted by our framework are as follows:

- As RBAC has become the de facto access control standard for organizations [32], [40], we assume that the cloud provider employs this (RBAC) model with hybrid hierarchy for implementing collaborating domain policies.
- Cloud vendor provides mechanism to map the requested permissions into a set of roles.

The framework consists of three major modules: *(i) collaboration request processing, (ii) conflict detection, and (iii) conflict removal*. The collaboration request processing module generates role sequences corresponding to interoperation requests. The conflict detection module enables a participating domain to detect if any role sequence is safe or unsafe, locally. On detection of unsafe sequence, the conflict removal module prevents either junior roles to inherit permissions belonging to senior roles or, a user, who has previously activated a role, from acquiring permissions of a conflicting role.

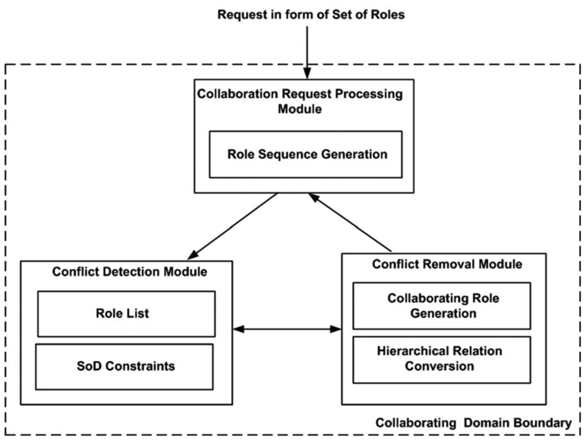

Request in form of Set of Roles
Collaboration Request Processing Module
Role Sequence Generation
Conflict Detection Module
Role List
SoD Constraints
Conflict Removal Module
Collaborating Role Generation
Hierarchical Relation Conversion
Collaborating Domain Boundary

Fig. 2. Distributed secure collaboration framework.

For any cross-domain interoperation, the request gets initiated from a role, and gets terminated at another one. We term them as *entry* and *exit* roles, respectively, following the conventions proposed in [15]. It is to be noted that entry and exit roles may coincide. Interoperation requests for which entry and exit roles are the same, we prefer to use it only once for ease of representation.

Formally, an interoperation request may be defined as follows:

Definition 1 (Interoperation Request). *If I is a set of interoperation requests then any request $I_i \in I$ made by an intermediate requester domain Q_i towards an intermediate provider domain P_i after being processed is given as: $I_i = \{r_Q^X, r_P^E\}$, where $r_Q^X \in Q_i$, $r_P^E \in P_i$ are the exit and entry roles activated in corresponding domains. If the request is processed partially, then currently, the framework has only activated the exit role from requester domain.*

In the above definition, processing an interoperation request includes determining if the request is authentic, as well as mapping the permissions into local roles for subsequent activation. On contrary, a partially processed request implies that the permissions which have been requested in the local domain have been found to be legitimate, however, a subset of mapped roles is not yet activated.

A role sequence generated from the set of interoperation request I is an alternative occurrences of entry and exit roles from multiple domains. It can be defined as following:

Definition 2 (Role Sequence). *A role sequence S generated from an interoperation request set I is defined as an ordered succession of entry and exit roles acquired by a user during a given session from a requester domain Q to a provider domain P. It has the following properties:*

1) $S = r_1, r_2, \dots r_n$ *where* $r_1 = r_Q^E, r_2 = r_Q^X, \dots, r_n = r_P^X$
2) $\forall (i < j), (r_i, r_j) \in S, if\ Dom(r_i) = Dom(r_j) \Rightarrow (r_i > r_j)$
3) *If the present interoperation request I_i is partially processed, it generates a partial role sequence $S'_i : S'_i = S_{i-1} \cup r_{Q'}^X$, where $r_{Q'}^X \in Q'$ is the exit role in current domain Q'*

We use $>$ symbol to denote dominance relation between a pair of roles, belonging to the same domain.

A role sequence is ordered because each role in the sequence will be acquired only after the previous role has been activated. If any domain has same entry and exit roles, we will use the notation of exit role in the role sequence.

We distinguish between safe and unsafe role sequences on the basis of secure interoperation. Principle of security is ensured if in a federated system $Q = \langle V, A \rangle$, interoperation takes place securely. Here, $V = \bigcup V_i$ and $A = \bigcup (A_i \cup F) - R$, where V_i is vertex set representing roles of domain i, and arc set A_i represents the dominance relationship between the roles [8], [15]. For example, if role r_1 dominates role r_2, then $r_1 > r_2$ and $r_1, r_2 \in A_i$. F is defined as the set of cross-links given by a pair of roles belonging to two different domains, while R defines a set of restricted access on $\bigcup V_i$ such that $(u, v) \in R$, $u \in V_i$, $v \in V_j$ and $i \neq j$, these edges are prohibited during interoperation. Formally, secure interoperation is defined as [8]:

Definition 3 (Secure Interoperation). *Given $G_i = \langle V_i, A_i \rangle$, $i = 1, \ldots, n$. $Q = \langle \bigcup V_i, B \rangle$ is a secure interoperation if (i) $B \cap R = \phi$, and (ii) $(u, v) \in \bigcup V_i$, (u, v) is legal in A_i if and only if (u, v) is legal in B.*

In this work, we assume that all interoperation requests honor restricted access between constituent domains and therefore satisfies the first requirement. The second requirement ensures the principle of security, which has been defined in Section 2. In Fig. 1a, we observe that if the three interoperation requests are made simultaneously, cyclic inheritance conflict occurs due to generation of the following role sequence: $\langle Viewer_{D3}, Editor_{D1}, Editor\_1_{D2}, Editor_{D3}, Viewer_{D3} \rangle$. This role sequence violates the principle of security in $D3$, where a junior role (i.e. $Viewer_{D3}$) gains the permissions of a senior role (i.e. $Editor_{D3}$), which is otherwise prohibited in normal situation. However, if in Fig. 1a, *Interoperation 3* was made to $Viewer_{D3}$, even then a cycle would have been generated of the following form: $\langle Viewer_{D3}, Editor_{D1}, Editor\_1_{D2}, Viewer_{D3} \rangle$. In contrast, this role sequence does not violate the principle of security in $D3$. Hence, it can be concluded that mere presence of a cycle in role sequence does not convey the occurrence of inheritance conflict. Formally, we define a cycle (or, role cycle) as:

Definition 4 (Role Cycle). *Let $S = \langle r_1, r_2, \ldots, r_n \rangle$ be a role sequence. If $C \subseteq S$ and $C = r_i, r_{i+1}, \ldots, r_k$, then S contains a role cycle C if and only if $r_i = r_k$.*

Given the definition of role cycle, we distinguish between its two variants as follows:

Definition 5 (Safe Role Cycle). *A role cycle C is safe, if it does not violate the principle of security.*

Definition 6 (Unsafe Role Cycle). *A role cycle C is unsafe, if it violates the principle of security.*

If the role sequence is itself a cycle i.e. $S = C$, we will interchangeably use the terms *role sequence* and *role cycle* to denote the ordered succession of entry and exit roles. Inheritance and SoD violation conflicts can be prevented if unsafe role cycle is detected and resolved on-the-fly. In the next section, we discuss about finding the unsafe role cycle which leads to detection of access conflicts.

4.2 Detection of Unsafe Role Cycle

In Section 2.4, we discussed about authentication mechanism followed in Google Cloud Storage. Although this mechanism ensures that resources will be accessed by legitimate users, it does not remember whether that user has already acquired permissions which are conflicting to present ones. Our work addresses this issue and proposes a methodology to detect conflicts dynamically, and then remove it without preventing collaboration.

4.2.1 Detection of Cyclic Inheritance Conflict

As mentioned in Section 4.1, each interoperation request consists of a set of exit and entry roles from a requester to a provider domain, respectively. Based on the complexity of the request, there may be multiple entry/exit roles belonging to one or more domains. Role cycles are primarily generated due to the presence of multiple such roles from the same domain. Our focus is to detect *if the present role sequence has any possibility to form an unsafe role cycle in near future*. Unauthorized access due to inheritance conflicts can only be prevented if it is detected prior to activation of the roles in the local domain. Therefore, to detect whether the interoperation request will generate an unsafe role cycle in future, our framework enables domains to maintain a list of local entry and exit roles which have been activated to allow previous interoperations, initiated during a particular user's session. For each new entry role requested, the framework checks if this role is *senior* to at least one previously visited (activated) roles in the list. On acquiring such a role, the framework concludes that if the current entry role is allowed to activate, it will generate an unsafe role cycle leading to violation of the principle of security.

The inheritance conflict detection rule is formally defined as:

Definition 7 (Inheritance Conflict Detection Rule). *Let $I = \{I_1, I_2, \ldots, I_n\}$ be a set of n interoperation requests which are made in a given user session, among which $(i - 1)$ of them have been processed. S_{i-1} is the corresponding ordered role sequence generated before activating the next role r_D^E in domain D, $S'_i = S_{i-1} \cup r_M^X$ is the partial sequence, M being the last visited domain. S'_i will generate an unsafe role cycle iff the following conditions are satisfied:*

- *C1. (Necessary condition) R_D is a non-empty set of previous activated roles in D: $R_D \neq \phi$*
- *C2. (Sufficient condition) $\exists r \in R_D : r_D^E > r$.*

Theorem 1 proves that the above rules assure detection of inheritance conflict if all conditions ($C1$ and $C2$) are verified before a role is added to the sequence.

Theorem 1. *Let S_i be a role sequence, and $S'_{i+1} = S_i \cup r_D^E$ be a partial sequence that satisfies the inheritance conflict detection rules. Then, S'_{i+1} will generate an unsafe role cycle.*

Proof. We assume that initial role sequence $S_i = r_1, r_2, \ldots, r_i, \ldots, r_n$ does not generate an unsafe role cycle, where $r_1 = r_H^E$, $r_2 = r_H^X$, $r_n = r_M^X$, H is the original requester domain and M is any intermediate domain, whose request has been already processed. It is assumed that the current provider domain D has been visited earlier. We proceed using a proof by contradiction. Assume to the contrary that the updated sequence S_{i+1} will not generate an unsafe role cycle after it has activated the entry role r_D^E in D. In this situation, no violation is created by S_{i+1}. As the current domain D has been visited earlier, there exists a non-empty set R_D containing previous entry and exit roles. Hence, the necessary condition ($C1$) of the inheritance conflict detection rule is satisfied. As S_{i+1} is a role sequence, by Definition 2, R_D contains previously activated roles in D in ordered sequence i.e. $r_j > r_{j+1} > \cdots > r_i > \cdots > r_D^E$, r_D^E being the last activated role in the sequence. In the role hierarchy of domain D, we assume $r_D^E > r_i$ which satisfies the sufficient condition ($C2$) of the inheritance conflict detection rule. Now, we have two cases: (i) $r_j > r_{j+1} > \cdots > r_i > \cdots > r_D^E$ and (ii) $r_D^E > r_i$ which contradicts each other. It implies that there exists a possibility of generation of an unsafe role cycle if both necessary and sufficient conditions of the conflict detection rule are satisfied. Hence, this is a contradiction to our earlier assumption that S_{i+1} will not form any unsafe role cycle. □

4.2.2 Detection of SoD Constraint Violation

SoD policies have been found to be crucial for securing many commercial and business applications. It reduces the possibility of fraud by partitioning of tasks and associated privileges [41]. In [33], four SoD models have been identified:

1) *Role-specific SoD*. A role-specific SoD disallows activation of conflicting roles by the same user in the same session or in concurrent sessions.
2) *Permission-specific SoD*. A permission-specific SoD prevents the same user to access conflicting permissions in the same session or in concurrent sessions.
3) *Role-level-user-specific SoD*. A role-level-user-specific SoD prohibits conflicting users of a role from assuming that role in concurrent sessions.
4) *Permission-level-user-specific SoD*. A permission-level-user-specific SoD prevents conflicting users of a permission from accessing that permission concurrently.

Violations of SoD constraints may occur during interoperation because of interplay of various policy constraints across domains. In this work, we focus on dynamic detection and removal of violation of *role-specific SoD*, which is a convenient way for expressing and enforcing constraints in role-based environment.

With reference to the above definition of role-specific SoD, two roles are termed to be *conflicting* if the following definition is valid:

Definition 8 (Conflicting Role Pair). *If any domain D contains two roles r_i, r_j, then the pair (r_i, r_j) is conflicting, if the following conditions are satisfied:*

- *L1. Role pair (r_i, r_j) does not have any dominance (hierarchical) relationship between each other.*
- *L2. Concurrent or simultaneous activation of (r_i, r_j) by a particular user in a given session is restricted by SoD constraint.*

In the proposed framework, the conflict detection module enables a participating domain to specify lists of conflicting role pairs between which SoD constraints have been enforced (refer Fig. 2). While processing the current request, we check if the present entry role in local domain forms a conflicting pair with at least one role belonging to its SoD constraint list. On obtaining such a role, the framework concludes that if the current entry role is allowed to activate, it will generate a violation of local SoD constraint. We formally define this SoD violation detection rule as follows:

Definition 9 (SoD Violation Detection Rule). *Let $I = \{I_1, I_2, \ldots, I_n\}$ be a set of n interoperation requests which are made in a given user session, among which $(i-1)$ of them have been processed. S_{i-1} is the corresponding ordered role sequence generated before activating the next role in domain D, $S'_i = S_{i-1} \cup r_M^X$ is the partial sequence, M being the last visited domain. S'_i will generate a SoD violation iff the following conditions are satisfied:*

- *C1. (Necessary condition) R_D is a non-empty set of previous activated roles in $D : R_D \neq \phi$*
- *C2. (Sufficient condition) $\exists r \in R_D : (r_D^E, r)$ forms a conflicting role pair*

Theorem 2 proves that the above rule assures detection of SoD constraint violation if all conditions ($C1$ and $C2$) are true before a role is added to the sequence.

Theorem 2. *Let S_i be a role sequence, and $S'_{i+1} = S_i \cup r_D^E$ be a partial sequence that satisfies the SoD violation detection rule. Then, S'_{i+1} will generate an unsafe role cycle.*

Proof. We assume that initial role sequence $S_i = r_1, r_2, \ldots, r_n$ does not generate an unsafe role cycle, where $r_1 = r_H^E$, $r_2 = r_H^X$, $r_n = r_M^X$, H is the original requester domain and M is any intermediate domain, whose request has been already processed. It is assumed that the current provider domain D has been visited earlier. We proceed using a proof by contradiction. Assume to the contrary that the updated sequence S_{i+1} will not generate an unsafe role cycle after it has activated the entry role r_D^E in D. In this situation, no violation is produced by S_{i+1}. As the current domain D has been visited earlier, there exists a non-empty set R_D containing previous activated roles. Hence, the necessary condition ($C1$) of the SoD violation detection rule is satisfied. As S_{i+1} is a role sequence, by Definition 2, R_D contains previously activated roles in D in ordered sequence i.e. $r_j > r_{j+1} > \cdots > r_i > \cdots > r_D^E$, r_D^E being the last activated role in the sequence. Let r_i be a role in D such that $r_i \in R_D$ and (r_i, r_D^E) forms a conflicting pair in the role hierarchy. This satisfies the sufficient condition ($C2$) of the SoD violation detection rule. As (r_i, r_D^E) is a conflicting role pair, no dominance relation exists between r_D^E and r_i. Now, we have two cases: (i) $r_j > r_{j+1} > \cdots > r_i > \cdots > r_D^E$, (ii) $(r_D^E \not> r_i) \wedge (r_i \not> r_D^E)$ which contradicts each other. It implies that there exists a possibility of generation of an unsafe role cycle if both necessary and sufficient conditions of the conflict detection rule are satisfied. Hence, this is a contradiction to our earlier assumption that S_{i+1} will not form any unsafe role cycle. □

We present the algorithm to detect both inheritance conflict and SoD violation (refer to Algorithm 1). In this algorithm, we consider that domain D has initiated the conflict detection process. The partial role sequence (S'), the role list in D (R_D), current role to be activated in D (r_D^E), list of roles which has been previously detected to create conflicts in D (R_D^{conf}), such that $R_D^{conf} \subseteq R_D$, and the SoD constraints (C_D)

are provided as input. The output returned by the algorithm is whether there is a possibility of conflict (Boolean *True*) or not (Boolean *False*), if the current entry role (r_D^E) is activated in D. The outer *if-loop* implies the necessary condition, while the inner *if-loops* entail the sufficient conditions in Definitions 7 and 9.

Algorithm 1. Conflict Detection

```
Input: Partial role sequence ($S'$), Role list ($R_D$), Current entry
       role ($r_D^E$), Conflict generating role list ($R_D^{conf}$), SoD con-
       straints ($C_D$)
Output: $True$ or $False$
if $R_D \neq \phi$ then
    for $i = 1 \ldots |R_D|$ do
        if $r_D^E > r_i$ then
            break;
        end
        if $(r_D^E, r_i) \in C_D$ then
            break;
        end
    end
    $R_D^{conf} \leftarrow R_D^{conf} \cup r_i$;
    Return: $True$;
end
else
    $R_D \leftarrow R_D \cup r_D^E$;
    $S' \leftarrow S' \cup r_D^E$;
    Return: $False$;
end
```

In the next section, we present the approaches of removing these access conflicts by ensuring both security and fairness in collaboration.

5 Dynamic Removal of Conflicts

To ensure secure collaboration in cloud environment, we need to remove the detected unsafe role cycle, responsible for inheritance conflict and violation of SoD constraint. In previous approaches (available in the literature), for both standard as well as hybrid RBAC hierarchies, resolution of conflicts are done by restricting or removing one or more cross-domain links. These approaches, although ensures security, but have not been found to be fair to collaboration. In this work, we aim at preserving the principle of security without completely preventing interoperation among the participating domains.

5.1 Removal of Unsafe Role Cycle

As explained earlier, for loosely-coupled collaboration, interoperation request constitutes a set of permissions which are mapped into one or more roles in local domain. In some cases, exact mapping is available, while on other occasions, excess permissions are granted through allowing one or more roles. In the following section, we propose methods to resolve cyclic conflicts corresponding to two cases: (i) exactly matching role set; (ii) no-exact matching role set.

5.1.1 Removal of Cyclic Inheritance Conflict

In case (i), a conflicting entry role gets activated for processing the current interoperation request. Under such situation, the interoperation request to access the entry role should be prevented from inheritance. Hybrid hierarchy in RBAC model introduces *A-relation* between roles. Such an *activation* relation denotes that any user, who has been assigned a hierarchically senior role, can activate a junior role but cannot inherit its permissions, unless an explicit *I-relation* is also assigned [32]. Therefore, activation semantics prevents a user from inheriting permissions of a senior role, which became inevitable through cyclic inheritance conflict. Formally, we define the conflict removal rule for exactly matched role set as:

Definition 10 (Inheritance Conflict Removal Rule for Exactly Matched Role). *Let P_Q be a set of permissions requested by an exit role r_C^X from remote domain C to local domain D, such that $P_Q = Prm(r_D^E)$, Prm is permission set of $r_D^E \in D$. If r_D^E satisfies inheritance conflict detection rule with a previous role in R_D, then conflict removal is done by replacing the hierarchical relation $r_C^X >_{IA} r_D^E$ with $r_C^X >_A r_D^E$.*

For case (ii), exactly matched role from the given permission set cannot be formed. However, to honor availability issue of multi-party collaboration, let the role mapping scheme chooses a role whose permission set is maximally overlapped to the requested set. However, this may allow a user to inherit extraneous permissions of a hierarchically senior role in the same domain. Also, the user may activate conflicting roles which are subjected to SoD constraints. To remove conflicts under such circumstances, the conflict removal module introduces a new *collaborating (virtual) role* (refer to Figure B.1b of the *supplemental material, available online*). The collaborating role should have a set of permissions equivalent to the requested set and will be *I-junior* to that role in the local hierarchy, whose permission set is maximally overlapped with the requested ones. Formally, a collaborating role in a domain can be defined as follows:

Definition 11 (Collaborating Role). *Let P_Q be a set of permissions requested by an exit role r_C^X from a remote domain C to a local domain D for interoperation, such that $P_Q \subseteq Prm(r_D^E)$, Prm is permission set of $r_D^E \in D$. If r_D^E satisfies inheritance conflict detection rule with a previous role in R_D, a collaborating role r_D^{coll} is introduced which has the following properties:*

- *P1. $(r_C^X >_{IA} r_D^{coll}) \wedge (r_D^E >_I r_D^{coll})$*
- *P2. No dominance relation exists between r_D^{coll} and r_D^X*
- *P3. $Prm(r_D^{coll}) = P_Q$.*

From the above definition, it is clear that now the interoperation request is made to the collaborating role instead of the actual local role to remove conflicts. Incidentally, the collaborating role has no dominance/hierarchical relationships with previous exit role (r_D^X) in the domain. We now formally define the conflict removal rule for no-exact matching role as:

Definition 12 (Inheritance Conflict Removal Rule for No-Exactly Matched Role). *Let P_Q be a set of permissions requested by an exit role r_C^X from a remote domain C to a local domain D for interoperation, such that $P_Q \subseteq Prm(r_D^E)$, Prm is permission set of $r_D^E \in D$. If r_D^E satisfies inheritance conflict detection rule with a previous role in R_D, inheritance of extraneous privileges is prevented by introducing a collaborating role r_D^{coll}, such that $(r_C^X >_{IA} r_D^{coll}) \wedge (r_D^E >_I r_D^{coll})$.*

Application of the rules defined in Definitions 10 and 12 have been shown in Figure B.1 of the *supplemental material, available online*. Algorithm 2 describes the inheritance conflict removal mechanism in form of pseudo-codes. In this algorithm, we consider that the requesting domain is C and the resource providing domain is D. The input to the algorithm are the requested permission set (P_Q) from C, current exit role (r_C^X) from C, current entry role (r_D^E) from D, list of roles which has been previously detected to create conflicts in D (R_D^{conf}), and the permission set of r_D^E ($Prm(r_D^E)$). The $ifPresent$ method checks if role r_D^E is contained in R_D^{conf}. For exactly matched role sets P_Q and $Prm(r_D^E)$, we transform the IA-relation to the A-relation. Otherwise, a collaborating role is created by invoking *generateCollaboratingRole* method on permission set P_Q and $(r_C^X >_{IA} r_D^E)$ relation is converted to $(r_C^X >_{IA} r_D^{coll}) \wedge (r_D^E >_I r_D^{coll})$.

Algorithm 2. Inheritance Conflict Removal

```
Input: Requested permission set ($P_Q$), Current exit role ($r_C^X$),
       Current entry role ($r_D^E$), Conflict generating role list
       ($R_D^{conf}$), Permission set of $r_D^E$ ($Prm(r_D^E)$)
Output: Transformed hierarchical relationships
Initialize: $X, Y$;
$X \leftarrow if\ Present(r_D^E, R_D^{conf})$;
while $X == True$ do
  $Y \leftarrow if\ Equal(P_Q, Prm(r_D^E))$;
  if $Y == True$ then
    Transform: $(r_C^X >_{IA} r_D^E) \rightarrow (r_C^X >_A r_D^E)$;
  end
  else
    Initialize: $r_D^{coll}$;
    $r_D^{coll} \leftarrow generateCollaboratingRole(P_Q)$;
    Transform: $(r_C^X >_{IA} r_D^E) \rightarrow ((r_C^X >_{IA} r_D^{coll}) \wedge (r_D^E >_I r_D^{coll}))$;
  end
end
```

5.1.2 Removal of SoD Constraint Violation

For case (i), the removal mechanism of SoD constraint violation is similar to that of the cyclic inheritance one, i.e. to transform a hierarchical *IA-relation* with a *A-relation*. For no-exact matching role (case (ii)), a collaborating role has to be defined to mediate SoD violation (defined in Section 5.1.1). However, in this case, the collaborating role may contain permissions which are contradictory to the present entry role, which in turn, has SoD constraints with a previous exit role in local domain. In that case, it is essential to determine the contradictory permissions between the sets $Prm(r_D^E)$ and $Prm(r_D^{coll})$ and eliminate them from the collaborating role to remove violation of SoD constraint. We define contradictory permissions as:

Definition 13 (Contradictory Permissions). *Let $O_1, O_2, \ldots, O_m$ be objects and $A_1, A_2, \ldots, A_n$ be access modes in any domain D, such that $A_1 \geq A_2 \geq \cdots \geq A_n$. Two permissions $p_i \in Prm(r_D^E)$, $p_j \in Prm(r_D^{coll})$, are termed to be contradictory if the following conditions are satisfied:*

- *C1. $Role(p_i), Role(p_j) \in D$*
- *C2. If $A_i \in p_i,\ A_j \in p_j : (A_i \leq A_j) \vee (A_j \leq A_i) \Rightarrow A_i \neq A_j$*
- *C3. If $O_i \in p_i, O_j \in p_j$: $O_i = O_j$*

Therefore, as evident from the Definition 13, conflict arises if we have two roles from the same domain, whose permissions have the same object, whereas access mode in one of them dominates that in the other. Removal of violation of SoD constraint is done by eliminating the contradictory permissions from the permission set of collaborating role. Formally it can be defined as:

Definition 14 (SoD Conflict Removal Rule). *If $p_i \in Prm(r_D^E)$, $p_j \in Prm(r_D^{coll})$ are contradictory permissions, then the SoD conflict between the roles is removed by modifying the permission set of r_D^{coll} as: $Prm(r_D^{coll})' = Prm(r_D^{coll}) \setminus \{p_j\}$.*

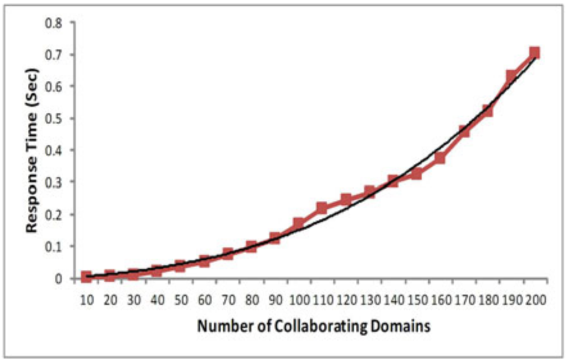
Application of SoD conflict rule has been shown in Figure C.1 of the *supplemental material, available online*. We explain the SoD conflict removal mechanism in Algorithm 3. Like the previous algorithm, we consider that the requesting domain is C and the resource providing domain is D. The inputs to this algorithm are same as those for Algorithm 2. If $r_D^E \in R_D^{conf}$, we check if its permissions are similar to the requested ones. If found equal, we simply transform the IA-relation to the A-relation. Otherwise, all contradictory permission pairs are determined from two sets ($Prm(r_D^E)$ and P_Q), and removed from P_Q. A collaborating role (r_D^{coll}) is generated by invoking the *generateCollaboratingRole* method on modified P_Q. Finally, making this collaborating role as mediator, the hierarchical relationship is transformed from $(r_C^X >_{IA} r_D^E)$ to $((r_C^X >_{IA} r_D^{coll}) \wedge (r_D^E >_I r_D^{coll}))$.

Algorithm 3. SoD Conflict Removal

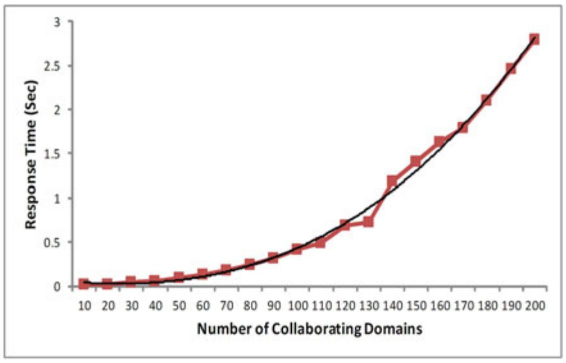
```
Input: Requested permission set ($P_Q$), Current exit role ($r_C^X$),
       Current entry role ($r_D^E$), Permission set of $r_D^E(Prm(r_D^E))$,
       Conflict generating role list ($R_D^{conf}$)
Output: Transformed hierarchical relationships
Initialize: $X, Y$;
$X \leftarrow if\ Present(r_D^E, R_D^{conf})$;
while $X == True$ do
  $Y \leftarrow ifEqual(P_Q, Prm(r_D^E))$;
  if $Y == True$ then
      Transform: $(r_C^X >_{IA} r_D^E) \rightarrow (r_C^X >_A r_D^E)$;
  end
  else
    for $i = 1 \ldots |Prm(r_D^E)|$ do
      for $j = 1 \ldots |P_Q|$ do
        Get all $(p_i, p_j)$ which are contradictory;
      end
    end
    $P_Q \leftarrow P_Q \setminus \sum p_j$;
    Initialize: $r_D^{coll}$;
    $r_D^{coll} \leftarrow generateCollaboratingRole(P_Q)$;
    Transform: $(r_C^X >_{IA} r_D^E) \rightarrow ((r_C^X >_{IA} r_D^{coll}) \wedge (r_D^E >_I r_D^{coll}))$;
  end
end
```

6 Results and Discussion

As explained in Sections 4 and 5, our objective is to ensure secure and fair collaboration among independent domains in cloud environment. Owing to highly dynamic nature of cloud, where the networks, services, requests, and the collaborators change frequently, it is imperative



(a) Time taken for Detection and Removal of Cyclic Conflict



(b) Time taken for Detection and Removal of Violation of SoD conflict

Fig. 3. Time to detect and remove access conflicts for variable number of domains.

that cloud service detects and removes such access conflicts efficiently. This is because, any cloud service provides scalability so that it can serve multiple requests simultaneously with minimum latency. In this section, we study the performance of our proposed conflict detection and removal schemes in terms of response time. We measure the efficiency of the proposed framework through two parameters: (i) number of participating domains among which collaboration is taking place, and (ii) size of role hierarchy (in terms of number of roles) of participating domains. The conflict detection and removal modules have been implemented in *JAVA* programming language. Collaborating domains and corresponding role hierarchies have been simulated to analyze the results based on the above mentioned parameters.

Domains, *roles* and *permissions* have been implemented as separate *JAVA* classes. Any permission has an *object* as well as an *access mode*, and a role has a *set of permissions*. There exists a role hierarchy in each of the participating domains. The role hierarchy is implemented using adjacency list data-structure. The conflict detection module has two array lists: (i) activated roles (initially empty), and (ii) role pairs on which predefined SoD constraints have been imposed. Each time a collaboration request from one domain to another is processed, activated role list is populated with new entry and exit roles in requesting and providing domain's array list, respectively. The conflict removal module has methods to check if any requested role is senior to any role in the activated list or has predefined SoD constraints. If such checks succeed, depending on the requested permission set, the conflict removal module either generates a collaborating role, or transforms relationships in the role hierarchy to restrict unauthorized access.

6.1 Response Time for Variable Number of Domains

In this section, we present the performance of our approach in a scenario where increasing number of participating domains gets involved in cloud-centric collaborating. The number of participating domains ranges from 10 to 200 in our simulation. We have assumed that the role hierarchy in all these domains is constant and contains ten roles. Interoperation requests form a cycle starting from first domain, involving other domains exactly once and then ending on the same domain where it started. The objective of this experiment is to study the performance of our approach when the number of participating domain gradually increases. We have used *System.nanoTime()* method in *java.lang.System* library to measure the time taken by our algorithm. For convenience of representation, we have plotted the results in terms of seconds in Fig. 3.

6.1.1 Discussion

To identify the nature of plots in Figs. 3a and 3b, we attempted to fit them against polynomial curves of varying degree. It is observed that best fit occurs for the polynomial of degree 4. It may be observed that for larger number of collaborating domains, time required for SoD conflict detection and removal (refer to Fig. 3b) is on the higher side in comparison to that of inheritance conflict. This is due to additional time required for checking if there exists (i) any predefined SoD constraint between a role pair under contention and (ii) conflicting permissions between an entry role and a collaborating role. From the nature of the curves, we claim that time required to detect and remove access conflicts is $O(n^4)$, where n is the number of participating domains.

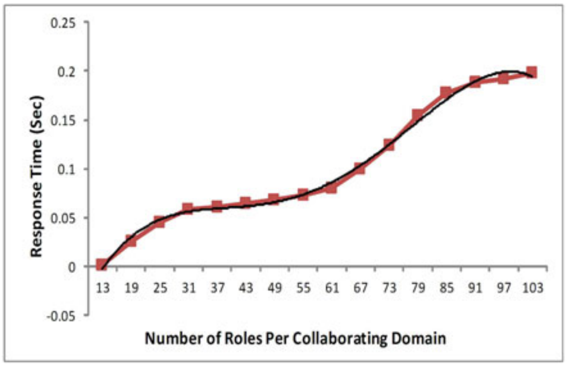
6.2 Response Time for Variable Size of Role Hierarchies

In this section, we assume five participating domains are involved in cloud supported collaboration, however, the size of the role hierarchy varies from 7 to 110. The objective of this experiment is to study the performance of our approach as the complexity of role hierarchy in the participating domain increases. Similar approach has been followed to measure the response time of our algorithm (refer to Fig. 4).

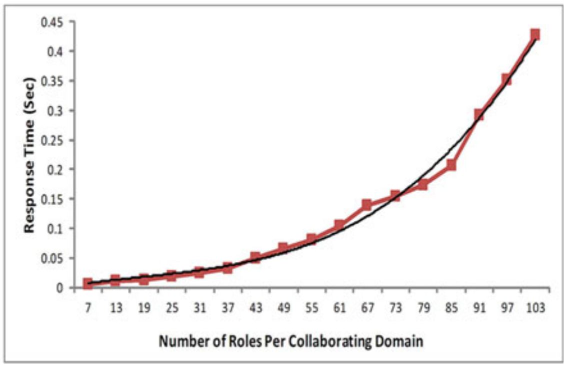
6.2.1 Discussion

It is evident from Figs. 4a and 4b, polynomial curve of degree 4 fits best for the generated curves. Similar to Fig. 3, we find that the time to detect and resolve SoD violation conflict is relatively higher compared to that for inheritance conflict. Hence, if a domain's role hierarchy consists of m roles, we claim that the time required to detect and remove access conflicts is $O(m^4)$.

In situations where both the number of domains and size of role hierarchy vary, the time taken to detect and remove



(a) Time taken for Detection and Removal of Cyclic Conflict



(b) Time taken for Detection and Removal of Violation of SoD conflict

Fig. 4. Time to detect and remove access conflicts for variable size of role hierarchy.

conflicts, in worst case, is $O((max\{n,m\})^4)$. Therefore, we conclude that our approach detects and removes access conflicts in polynomial time, addressing the scalability requirement of cloud-based collaboration service.

7 Conclusion

In present SaaS clouds, online collaboration is one of the popular offerings. However, ensuring secure and fair collaboration among participating domains is a challenging task. Owing to loosely-coupled nature of cloud-based collaboration, interoperation requests from a remote user are sent in form of a set of permissions. Provider domain maps these permissions into a set of local roles which are activated by the requester to accomplish tasks. Activation of multiple simultaneous roles during a particular user's session may introduce two types of access conflicts: (i) cyclic inheritance, and (ii) SoD constraint violation. These conflicts generate cycles which allow a user, already enabling a junior role, to inherit permissions of a senior or a constraint-restricted role and violate the principle of security. Dynamic and distributed detection of conflict has not been addressed in the literature. Moreover, reported works on removal of conflicts primarily deals with discarding the conflicting interoperation request. In this work, we propose a distributed secure collaboration framework for cloud collaboration service. It uses only local information to detect conflicts and remove them. The proposed approach is found to scale in polynomial time (of the order of 4) with respect to varying number of domains and roles, thus conforming to the scalability requirement of cloud-based services. In future, the proposed access conflict mediation approach will be extended to multi-cloud federated environment, where the issue of semantic heterogeneity will be addressed along with issues of security and availability.

Acknowledgments

The authors would like to thank the anonymous reviewers for careful reviews and valuable suggestions that helped in improving the quality of the manuscript. The work of N. Ghosh was partially supported by TCS Research Scholarship grant. The work of S. K. Das was supported by the US National Science Foundation grants under award numbers CNS-1355505, IIS-1404673 and CNS-1404677.

References

[1] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf, "NIST cloud computing reference architecture," *NIST Spec. Publ.*, vol. 500, p. 292, 2011.
[2] D. Banks, J. S. Erickson, and M. Rhodes, "Toward cloud-based collaboration services," in *Proc. Usenix Workshop HotCloud*, 2009.
[3] H. Li, B. Wang, and B. Li, "Oruta: Privacy-preserving public auditing for shared data in the cloud," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 43–56, Apr. 2014.
[4] N. Ghosh, S. K. Ghosh, and S. K. Das, "Selcsp: A framework to facilitate selection of cloud service providers," *IEEE Trans. Cloud Comput.*, vol. PP, no. 99, p. 1, Jul. 2014.
[5] J. B. D. Joshi, R. Bhatti, E. Bertino, and A. Ghafoor, "Access-control language for multidomain environments," *IEEE Internet Comput.*, vol. 8, no. 6, pp. 40–50, Dec. 2004, doi: 10.1109/MIC.2004.53.
[6] Y. Zhang, "An access control and trust management framework for loosely-coupled multi-domain environment," Ph.D. dissertation, School Inf. Sci., Univ. Pittsburgh, Pittsburgh, PA, USA, 2011.
[7] B. Shafiq, J. B. D. Joshi, E. Bertino, and A. Ghafoor, "Secure interoperation in a multidomain environment employing rbac policies," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 11, pp. 1557–1577, Nov. 2005, doi: 10.1109/TKDE.2005.185.
[8] L. Gong and X. Qian, "Computational issues in secure interoperation," *IEEE Trans. Softw. Eng.*, vol. 22, no. 1, pp. 43–52, Jan. 1996, doi: 10.1109/32.481533.
[9] D. E. Denning, "A lattice model of secure information flow," *Commun. ACM*, vol. 19, no. 5, pp. 236–243, 1976.
[10] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The nist model for role-based access control: Towards a unified standard," *ACM Trans. Inf. and Syst. Secur.*, vol. 4, no. 3, pp. 224–274, Aug. 2001.
[11] J. W. Gray, "Toward a mathematical foundation for information flow security," *J. Comput. Secur.*, vol. 1, no. 3, pp. 255–294, 1992.
[12] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Comput.*, vol. 29, no. 2, pp. 38–47, Feb. 1996, doi: 10.1109/2.485845.
[13] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Trans. Inf. and Syst. Secur.*, vol. 4, no. 3, pp. 224–274, 2001.
[14] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proc. IEEE*, vol. 63, no. 9, pp. 1278–1308, Sep. 1975, doi: 10.1109/PROC.1975.9939.
[15] M. Shehab, A. Ghafoor, and E. Bertino, "Secure collaboration in a mediator-free distributed environment," *IEEE Trans. Parallel and Distrib. Syst.*, vol. 19, no. 10, pp. 1338–1351, Oct. 2008.
[16] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati, "An algebra for composing access control policies," *ACM Trans. Inf. and Syst. Secur.*, vol. 5, no. 1, pp. 1–35, 2002.
[17] S. Dawson, S. Qian, and P. Samarati. (2002). Providing security and interoperation of heterogeneous systems. *Distrib. Parallel Databases*, vol. 8, pp. 119–145, 10.1023/A:1008787317852. [Online]. Available: http://dx.doi.org/10.1023/A:1008787317852
[18] B. Shafiq, J. S. Vaidya, A. Ghafoor, and E. Bertino, "A framework for verification and optimal reconfiguration of event-driven role based access control policies," in *Proc. 17th ACM Symp. Access Control Models and Technol.*, 2012, pp. 197–208.

[19] R. Bhatti, B. Shafiq, E. Bertino, A. Ghafoor, and J. B. Joshi, "X-gtrbac admin: A decentralized administration model for enterprise-wide access control," *ACM Trans. Inf. and System Secur.*, vol. 8, no. 4, pp. 388–423, 2005.

[20] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proc. 17th IEEE Symp. Secur. Privacy*, Oakland, CA, USA, 1996, pp. 164–173, doi: 10.1109/SECPRI.1996.502679.

[21] M. Y. Becker and P. Sewell, "Cassandra: Distributed access control policies with tunable expressiveness," in *Proc. Fifth IEEE Int. Workshop Policies Distrib. Syst. Netw.*, 2004, pp. 159–168.

[22] N. Li, J. C. Mitchell, and W. H. Winsborough, "Design of a role-based trust-management framework," in *Proc. IEEE Symp. Secur. Privacy*, 2002, pp. 114–130.

[23] A. J. Lee, M. Winslett, J. Basney, and V. Welch, "Traust: A trust negotiation-based authorization service for open systems," in *Proc. Eleventh ACM Symp. Access Control Models and Technol.*, 2006, pp. 39–48.

[24] N. Li and J. C. Mitchell, "A role-based trust-management framework," in *Proc. DARPA Inf. Survivability Conf. Exposition,*, vol. 1, 2003, p. 201.

[25] Z. Lu, Z. Wen, Z. Tang, and R. Li, "Resolution for conflicts of inter-operation in multi-domain environment," *Wuhan University J. Nat. Sci.*, vol. 12, no. 5, pp. 955–960, 2007.

[26] E. Bertino, P. A. Bonatti, and E. Ferrari, "Trbac: A temporal role-based access control model," *ACM Trans. Inf. and Syst. Secur.*, vol. 4, no. 3, pp. 191–233, 2001.

[27] F. Cuppens, N. Cuppens-Boulahia, and M. B. Ghorbel, "High level conflict management strategies in advanced access control models," *Electron. Notes Theor. Comput. Sci.*, vol. 186, pp. 3–26, 2007.

[28] E. C. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," *IEEE Trans. Softw. Eng.*, vol. 25, no. 6, pp. 852–869, Nov. 1999.

[29] J. Chomicki, J. Lobo, and S. Naqvi, "Conflict resolution using logic programming," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 1, pp. 244–249, Jan. 2003.

[30] J. D. Moffett and E. C. Lupu, "The uses of role hierarchies in access control," in *Proc. Fourth ACM Workshop Role-based Access Control*, 1999, pp. 153–160.

[31] R. Sandhu, "Role activation hierarchies," in *Proc. third ACM Workshop Role-Based Access Control*, 1998, pp. 33–40.

[32] J. B. D. Joshi, E. Bertino, and A. Ghafoor, "Temporal hierarchies and inheritance semantics for GTRBAC," in *Proc. Seventh ACM Symp. Access Control Models Technol.*, 2002, pp. 74–83.

[33] R. Li, Z. Tang, Z. Lu, and J. Hu, "Request-driven role mapping framework for secure interoperation in multi-domain environments," *Comput. Syst. Sci. Eng.*, vol. 23, no. 3, pp. 193–206, 2008.

[34] A. Almutairi, M. Sarfraz, S. Basalamah, W. Aref, and A. Ghafoor, "A distributed access control architecture for cloud computing," *IEEE Softw.*, vol. 29, no. 2, pp. 36–44, Apr. 2012, doi: 10.1109/MS.2011.153.

[35] M. Singhal, S. Chandrasekhar, T. Ge, R. S. Sandhu, R. Krishnan, G.-J. Ahn, and E. Bertino, "Collaboration in multicloud computing environments: Framework and security issues" *IEEE Comput.*, vol. 46, no. 2, pp. 76–84, Feb. 2013.

[36] M. Almorsy, J. Grundy, and A. S. Ibrahim, "Collaboration-based cloud computing security management framework," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2011, pp. 364–371.

[37] H. Takabi and J. B. Joshi, "Policy management as a service: An approach to manage policy heterogeneity in cloud computing environment," in *Proc. IEEE 45th Hawaii Int. Conf. Syst. Sci.*, 2012, pp. 5500–5508.

[38] A. Gouglidis, I. Mavridis, and V. C. Hu, "Security policy verification for multi-domains in cloud systems," *Int. J. Inf. Secur.*, vol. 13, no. 2, pp. 97–111, 2014.

[39] J. M. A. Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray, "Toward a multi-tenancy authorization system for cloud services," *IEEE Secur. Privacy*, vol. 8, no. 6, pp. 48–55, Dec. 2010.

[40] S. Du and J. B. D. Joshi. (2008). Supporting authorization query and inter-domain role mapping in presence of hybrid role hierarchy. in *Proc. 11th ACM Symp. Access Control Models and Technologies*, ser. SACMAT '06, New York, NY, USA: ACM, pp. 228–236. [Online]. Available: http://doi.acm.org/10.1145/1133058.1133090

[41] G.-J. Ahn and R. Sandhu, "Role-based authorization constraints specification," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 207–226, 2000.

Nirnay Ghosh received the BTech degree in computer science and engineering from West Bengal University of Technology, Bidhan Nagar, India, and the MS degree (by research) in information technology from Indian Institute of Technology (IIT), Kharagpur. He is currently working toward the PhD from the School of Information Technology (SIT) under the IIT, Kharagpur, India. His broad area of research includes security in cloud computing, trustworthy selection of service provider for secure multi-domain interoperations. He is a student member of the IEEE and published six international conference papers, two journal papers, and one book chapter during the course of his MS program.

Debangshu Chatterjee received the BTech degree in computer science and engineering from West Bengal University of Technology, Bidhan nagar, India. He received the MTech from the School of Information Technology (SIT) under Indian Institute of Technology (IIT), Kharagpur, India, in 2014. His broad area of research is securing access control in cloud environment.

Soumya K. Ghosh received the MTech and PhD degrees in computer science and engineering from the Indian Institute of Technology (IIT) Kharagpur, India. He is currently a Professor at the School of Information Technology, IIT Kharagpur. Prior to joining IIT Kharagpur, he worked for Indian Space Research Organization in the area of Satellite Remote Sensing and GIS. His research interests include cloud computing, spatial information retrieval and knowledge discovery. He has published over 100 articles in journals and conference proceedings. He is a member of IEEE.

Sajal K. Das is currently the Chair of Department of Computer Science and the Daniel St. Clair Endowed Chair at the Missouri University of Science and Technology, Rolla, MO. Prior to that, he was a Distinguished Scholar Professor of Computer Science and Engineering and the Founding Director of the Center for Research in Wireless Mobility and Networking (CReWMaN) at the University of Texas at Arlington (UTA), TX. His current research interests include wireless and sensor networks, mobile and pervasive computing, cyber-physical security, distributed and cloud computing, biological and social networks, applied graph theory and game theory. He has published more than 500 technical papers and 47 invited book chapters in these areas, holds five US patents, and received nine Best Paper Awards in international conferences. He is a senior member of the IEEE.