COMPUTERS & SECURITY 28 (2009) 260–275

available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Towards secure dynamic collaborations with group-based RBAC model

Qi Li[a], Xinwen Zhang[b], Mingwei Xu[a,*], Jianping Wu[a]

[a]Department of Computer Science, Tsinghua University, Beijing 100084, China
[b]Samsung Information Systems America, San Jose, CA 95134, USA

ARTICLE INFO

Article history:
Received 22 April 2007
Received in revised form
20 November 2008
Accepted 12 December 2008

Keywords:
Access control
RBAC
Group-based RBAC
GB-RBAC
Secure collaborations

ABSTRACT

Role-Based Access Control (RBAC) has become a popular technique for security purposes with increasing accessibility of information and data, especially in large-scale enterprise environments. However, authorization management in dynamic and ad-hoc collaborations between different groups or domains in these environments is still an unresolved problem. Traditional RBAC models cannot solve this problem because they cannot support security policy composition from different groups, and lack efficient administrative models for dynamic collaborations. In this paper, we propose a group-based RBAC model (GB-RBAC) for secure collaborations which is based on RBAC96 and extended with group concept to capture dynamic users and permissions. We propose a decentralized security administrative model for GB-RBAC to address the management issues of RBAC in collaborations. As a unique property, our model supports two levels of authorization management: global or system level management by system administrators and local or group level management by group administrators. In this way, our model implements the principles of management autonomy and separation of duty (SoD) in security administrations. We apply our model for authorization management in collaborations by introducing the concept of virtual group. A virtual group is built for a collaboration between multi-groups, where all members build trust relation within the group and are authorized to join and perform operations for the collaborative work. Compared with existing work, our model supports dynamic and ad-hoc collaborations in large-scale systems with the properties of controllable, decentralized, and fine-grained security management.

1. Introduction

The past decade has seen the emergence and wide use of e-commerce and e-government systems, in which web and Internet-based services are becoming pervasive. More and more *collaborative* applications have been developed based on these existing systems and infrastructures to increase the efficiency and productivity. Typically, in a collaborative work, users from different groups cooperate by operating on some shared resources (Nita-Rotaru and Li, 2004). With the increasing scalability of *collaborations*, access and usage control of information and services becomes very complex. The fundamental security problem in a collaboration is to control the admission of users to collaborations, and their permissions to access the resources, typically based on their job duty or skills. Researchers have proposed and implemented many access control models. Among them, Role-based Access Control (RBAC) is the most attractive solution

* *Corresponding author*. Tel.: +8610 62785822.
E-mail addresses: liqi@csnet1.cs.tsinghua.edu.cn (Q. Li), xinwen.z@samsung.com (X. Zhang), xmw@csnet1.cs.tsinghua.edu.cn (M. Xu), jianping@cernet.edu.cn (J. Wu).
0167-4048/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2008.12.004

with the properties of "policy-neutral" and simplicity of administration. In a RBAC model, both permissions and users are assigned to roles by system administrators, such that a user obtains the permissions of the assigned roles.

Traditional RBAC models such as RBAC96 (Sandhu et al., 1996) and NIST RBAC standard (Ferraiolo et al., 2001) cannot provide efficient authorization management for collaborations. The main reason is that these models focus on controlling user permissions according to pre-assigned roles and permission-role assignment relations. While in dynamic environments, roles and user-role assignment relations are not fixed during collaborations. On the other side, because the numbers of roles and users in a RBAC system vary from tens to thousands in large enterprise systems, authorization management is a big obstacle for secure access. Many administrative models for RBAC have been proposed for convenience and management efficiency purposes (Crampton and Loizou, 2003; Crampton, 2005; Osborn et al., 2000; Sandhu et al., 1999; Oh et al., 2006). However, most (if not all) of these models define administration policies based on existing role hierarchies such that they cannot support dynamic and ad-hoc collaborations between groups since the role hierarchy in these scenarios is not static.

Many advanced RBAC models have been proposed for authorizations in multi-domain environments. For example, a policy composition framework is proposed in (Shafiq et al., 2005; Joshi et al., 2004; Piromruen and Joshi, 2005) to integrate RBAC policies from multiple domains and separation of duty (SoD) constraints are analyzed. Most of these recent work addresses secure collaborations using direct role mapping (Shafiq et al., 2005; Joshi et al., 2004; Kapadia et al., 2000) and can solve some problems when integrating RBAC policies, such as role promotion (Kapadia et al., 2000) and user-specific Separation of Duty (SoD) violation, role-specific SoD and role-assignment violations (Shafiq et al., 2005). However, in role mapping approach, a role in one group is mapped to a role in another group, which requires non-trivial efforts from security administrators for permission management, especially with hundreds or thousands of users, roles, and their inter-relationships, because in general only a small team of security administrators are delegated to manage these components. The problem becomes worse with collaborations where dynamic user-role and permission-role assignments are required. For example, different video conferences always need different users in a group to participate, so there must be many administrative tasks for administrators to dynamically modify role and permission assignments for video conferences. It is impossible and infeasible for few local administrators to perform these assignments. Therefore, the management issue of RBAC is an important factor which directly restricts its deployment and usage in dynamic collaborations.

Our motivation behind this issue is to simplify decentralize administrative tasks, and thus enhance the practicability of RBAC in dynamic collaboration environments. In this paper, we propose a truly decentralized and group-based RBAC (GB-RBAC) model by introducing the concept of groups and modifying the user-role assignment model in previous work (Oh et al., 2006; Sandhu et al., 1999). GB-RBAC model retains the main features of RBAC and simplifies user-role assignments with two mechanisms. First, GB-RBAC provides a default group role set (*DSet*) to reduce administrative tasks through the group component. In this way, a new member of a group can be assigned with a set of default roles without administrator's involvement. Secondly, a group administrator can assign other explicit roles to a group member based on local administrative policies in a fine-grained manner. Hence, simplified but flexible user-role assignment is implemented in our model. Therefore, our model provides a two-level administrative model to facilitate RBAC administrative issues: global or system level and local or group level. Thus the following advantages can be achieved.

- Our model naturally supports decentralized management in a simple and efficient way. For example, in group collaboration systems (Nita-Rotaru and Li, 2004), group administrators can add or modify assignments to meet the application and local administration requirements without global administrators' involvement. This simplifies system-level administrative tasks, and provides a flexible administration mechanism for dynamic user-role assignments, especially in ad-hoc collaboration environments.
- Our administrative model provides tunable group-level administrative permissions, which are controlled by system-level administrators. Therefore, not only is user-role assignment for system administrator greatly simplified, but the principle of separation of duty (SoD) in administrative level (Crampton, 2006) is also implemented.
- Our administrative model satisfies the requirements of autonomy administration for RBAC, such as fine-grained user-role assignment and tunable group-level administration (Nissanke and Khayat, 2004), which are not solved completely by previous work because of centralized administrative domain with few system-level administrators.

The contributions of this paper are four-fold.

(i) We propose a GB-RBAC model, which is based on the RBAC96 model and extended with a group concept. We identify the difference between our model and other group-based approaches.
(ii) We develop a two-level administrative model for GB-RBAC with the features of decentralized management, tunable group-level administrative permissions, and the principle of administrative SoD.
(iii) We develop a mechanism to use our model for secure ad-hoc collaborations between groups by introducing the concept of virtual group. We define algorithms to perform user-role and group-role assignments in virtual groups.
(iv) We develop an authorization service based on the GB-RBAC model, and the prototype of the model shows the feasibility in the real distributed applications. Our model provides acceptable performance for ad-hoc collaboration applications.

The rest of this paper is organized as follows. We discuss related work and the difference from our approach in Section 2. The GB-RBAC model and its administrative model are presented in Section 3 and Section 4, respectively. Section 5 explains the ad-hoc collaboration scheme with GB-RBAC. Section 6 presents

an overview of the implementation of our approach. Section 7 concludes this paper and presents our future work.

2. Related work

Sandhu et al. define a set of RBAC models (Sandhu et al., 1996; Sandhu et al., 1999). Afterwards, Ferraiolo et al. (2001) propose the NIST standard of RBAC. Different architectures for RBAC services on the web are proposed in Park et al. (2001). In these RBAC models, however, static user-role association is used, which is tedious to configure every user-role assignment. Moreover, in the user-role administrative model known as URA97 of ARBAC97 (Sandhu et al., 1999), multiple steps are needed to assign a user to a role because the prerequisite conditions of user-role assignment relations in URA97 are defined with regular roles, which form a role hierarchy (Sandhu et al., 1999). As a result, it is difficult to administrate the policies when the model is deployed in a large enterprise system because it has many groups or departments.

ARBAC02 (Oh et al., 2006) addresses the multiple user-role assignments and duplicated information problems by defining user-role relations based on existing organization structure information as user pools and permission pools, such as user's position from human resource department and permissions from IT department, instead of the regular roles. The limitation of this model is that roles in a user pool must have partial order relation. It results great constraints on user-role assignment in a situation where there are many diverse discrete roles. Another weakness in ARBAC02 is that it requires pre-defined user pools (OS-U) and permission pools (OS-P), and this introduces complex tasks for administrators.

Nyanchama and Osborn (1999) propose a role graph model which is equivalent with the role hierarchy in Sandhu's RBAC model. Instead of defining explicit user-role assignment relations, this model also introduces the concept of group to provide implicit user-role assignment (Nyanchama and Osborn, 1999; Osborn and Guo, 2000). Actually the group-role assignment is much similar with user-role assignment in original RBAC. So this model does not consider the vast administration tasks for administrators, and does not fully utilize the components of groups to facilitate administrative model. In addition, the administration of the role graph model is centralized, which offers less flexible than our approach. In GB-RBAC model, the *DSet* can support batch of user assignments automatically. Superficially, this mechanism is equivalent to that in the group graph model (Osborn and Guo, 2000).

Crampton et al. provide four types of role hierarchy administration based on a role graph to facilitate role administration (Crampton and Loizou, 2003; Crampton, 2005). Koch et al. (2004) extends this work to implement precise semantics and the systematic verification of constraints. However, these models heavily rely on the role hierarchy and focus on graph modification; that is, they do not consider user-role assignment. Based on Sandhu's administrative RBAC model (specifically, URA97 in ARBAC97; Sandhu et al., 1999), we address more complete decentralize management of user-role assignment, and our model solves the problem of access control in group communication applications by providing two ways of user-role assignments. Moreover, our model supports autonomy administration which provides an easy and versatile way for various static and dynamic assignments.

Basically, the ARBAC97 model (Sandhu et al., 1999) focuses on a centralized administrative model where a single super security officer (SSO) defines roles and conducts user-role and permission-role assignments. The ARBAC02 (Oh et al., 2006) leverages organization structures in an enterprise to simplify these tasks. Both models do not solve the problem in decentralized environment where flexible and autonomous authorization management is desired. Our two-layer administrative model improves the existing model on this aspect and provides flexible and scalable management in collaborative computing environments. Specifically, in system-level administrators define roles and permission-role assignments, while group-level administrators assign users roles based on local group policies. In this way, our model supports fine-grained administrations according to local administrative policies, which is referred as administration autonomy, a fundamental principal for RBAC administration (Sandhu et al., 1999).

Recently, several research efforts have been devoted to the topic of inter-operation in multi-domain environments (Shafiq et al., 2005; Joshi et al., 2004; Piromruen and Joshi, 2005; Kapadia et al., 2000). In (Kapadia et al., 2000), Kapadia et al. propose a dynamic role translation model, and several security issues are provided. Shafiq et al. (2005), Piromruen and Joshi (2005) and Joshi et al. (2004) propose a series of secure inter-operation schemes. In (Joshi et al., 2004), Joshi et al. propose an XML based RBAC to specify multi-domain policies. In (Shafiq et al., 2005; Piromruen and Joshi, 2005), solutions are proposed based on the Generalized Temporal Role-Based Access Control Model (GTRBAC). In (Shafiq et al., 2005), Shafiq et al. analyze three types of violations when integrating RBAC policies: user-specific separation of duty (SoD) violation, role-specific SoD violation, and role-assignment violation. For example, a role-assignment violation happens when a user of a domain is allowed to access a role even though the user is not directly assigned to the role or any of the roles that are senior to the role in the role hierarchy of the domain. In addition, Piromruen et al. transform local GTRBAC policy to facilitate inter-domain operations (Piromruen and Joshi, 2005). These approaches use bottom-up approach to composite RBAC policies and have to address many problems when emerging polices, such as role covert promotion[1] and all kind of violations above mentioned. Tolone et al. (2005) discuss access control requirements in collaborative systems and analyze existing access models including RBAC in collaborative environments. In our secure collaboration scheme, we propose a top-bottom approach to merge RBAC policies of different groups, thus our scheme avoids some problems introduced in role mapping approaches, such as role-specific SoD and role-assignment violations. The component of virtual group is introduced to avoid direct role mapping. In this way, most of problems mentioned above are eliminated.

[1] The covert promotion problem appears when a user crosses group boundaries and returns to a local group with a role senior to his original roles in the group (Kapadia et al., 2000).

3. The GB-RBAC model

3.1. Overview of GB-RBAC

The GB-RBAC model proposed in this paper introduces the concept of group, through which new user-role assignment mechanisms are built. The essential difference between groups and roles is that a group is a collection of users who have the similar security attributes, while a role is a collection of permissions (Sandhu, 1995)[2].

Fig. 1 demonstrates all the components in the GB-RBAC model and how it works. The users, roles, resources, operations, and permissions are similar to those in the RBAC96 (Sandhu et al., 1996) model. Typically, permissions associate operations on resources, and both permissions and users are assigned to roles. GB-RBAC introduces two new components: groups and group leaders (administrators). A group, as mentioned above, is a collection of users and a group administrator is a user assigned to a group administrative role. Besides assigning users to roles by system administrators, GB-RBAC supports user-role assignment by group administrators. In Fig. 1, there are two types of lines: the solid lines denote associations between components and the dashed lines denote management on these associations. Because of the new component of groups, we can easily find that the dashed lines among the groups, roles, permissions, and operations are not seen in previous models. Therefore, two levels of user administrations can be provided by GB-RBAC. The first is the system-level administration associated with centralized control over user-role assignment, that is denotes by the dashed lines whose start point is the node of the system administrator. The second is the group-level administration associated with decentralized control over user-role assignment, which is denoted by the dashed lines whose start point is the node of group leaders. In the rest of this paper, the two terms, system-level administrative model and group-level administrative model, refer respectively to these two administrative approaches.

3.2. Model description

GB-RBAC incorporates the component of groups into the RBAC96 (Sandhu et al., 1996) model and provides decentralized role administration. GB-RBAC indirectly imposes access control on a user's action after this user is authenticated and assigned to a set of roles by default. Fig. 2 shows the components of a GB-RBAC model. The concepts of users (U), roles (R), role hierarchy (RH), permissions (P), permission-role assignment (PA), and sessions (S) are identical to the original RBAC96 model (Sandhu et al., 1996). Besides these, GB-RBAC includes a set of groups (G). Each group is assigned with a set of roles (group-role assignment or GA). A user can belong to one or more groups, which is represented as the user-group mapping (UM). In addition, we propose two layers of roles which are referred as system-level roles (SR) and group-level roles (GR).

[2] Although a role is considered as a set of users and permissions in RBAC (Sandhu, 1995), for user-role administration purpose, we consider a role as a set of permissions here.

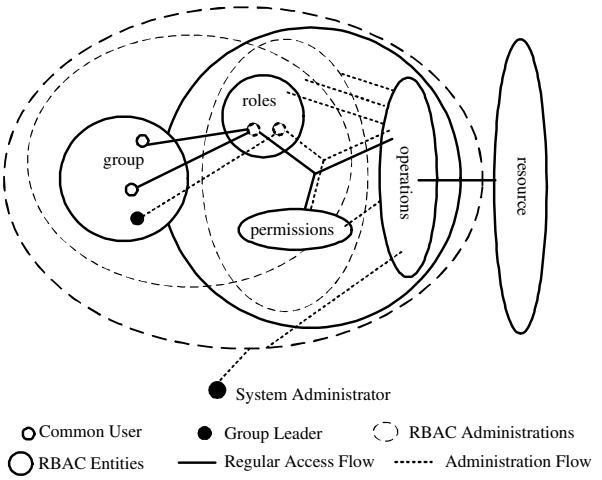

Fig. 1 – Overview of a GB-RBAC system.

Role hierarchy relationships (RH) are included in these two layers of roles like that in RBAC96. As Fig. 2 shows, RH exists in both level roles in GB-RBAC.

Besides the user-role assignment in system scope which is similar to the user-role assignment in URA97 (Sandhu et al., 1999), there is another type of user-role assignment which happens in group scope. Specifically, as UM associates users with groups and GA associates roles to groups, a group administrator can assign a user in UM to a role in GA, which is called group-level user-role assignment (GUA), while the original one is called system-level user-role assignment (SUA). In another word, GUA serves as the mechanism through which a role can be assigned to a user because the user has a mapping relation with a group and the role is assigned to the group, and then the user holds the permissions to access resources defined with the role. As Fig. 2 shows, GUA is built upon the relationship of UM and GA. That is, GUA is only effective if users are in the context of groups assigned with roles. Through the mechanism of mapping users to groups (UM) and assigning role to groups (GA), a new concept of default group roles set (*DSet*) is introduced in GB-RBAC, which indicates the set of roles assigned to a user in a group by default.

The formal definitions of individual components in GB-RBAC are defined as follows.

Definition 1. A GB-RBAC model has the following components:

- U, P, SR, GR, S, and G (users, permissions, system-level roles, group-level roles, sessions, and groups, respectively).
- $R = SR \cup GR$. For simplicity we assume $SR \cap GR = \emptyset$ in this paper.
- $PA \subseteq P \times R$, a many-to-many permission to role-assignment relation.
- $UM \subseteq U \times G$, a many-to-many user to group mapping relation. This relation shows that a user can be mapped into many different groups.
- $GA \subseteq G \times R$, a many-to-many group to role-assignment relation.

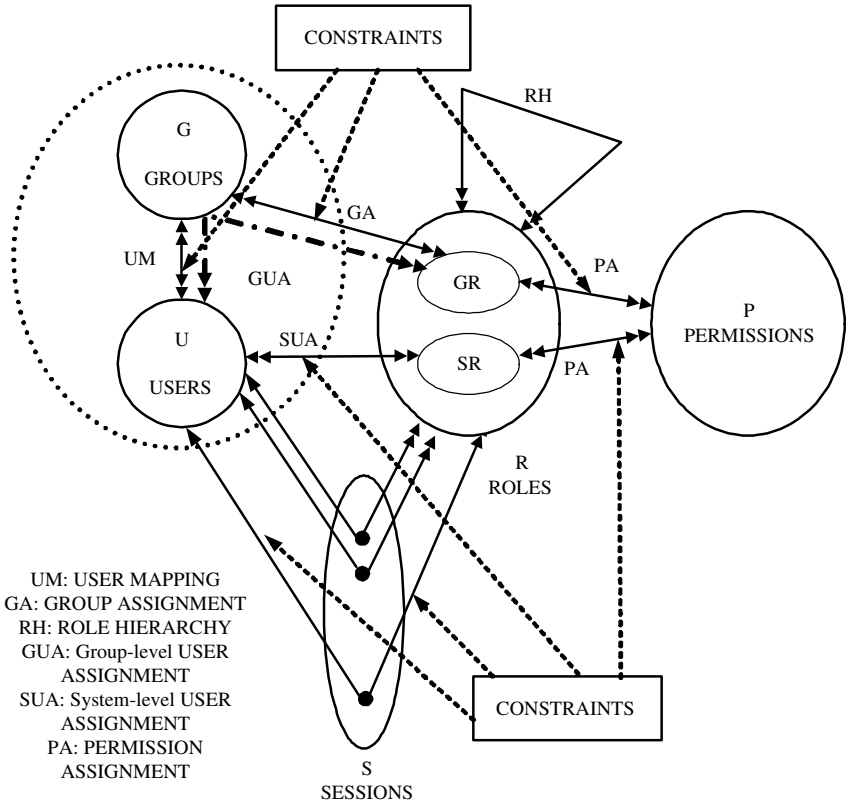

Fig. 2 – GB-RBAC model.

- SUA $\subseteq U \times SR$, system-level user-role assignment.
- GUA $\subseteq U \times GR$, group-level user-role assignment, and $(u, r) \in GUA$ only if $((u, g) \in UM) \wedge ((g, r) \in GA)$.
- UA = SUA ∪ GUA, a many-to-many user-role assignment relation.
- RH $\subseteq R \times R$, a partial order on R called the role hierarchy or role dominance relation. For any two roles r_1 and r_2, $r_1 \geq r_2$ means that r_1 has partial relation over r_2.
- user: $S \rightarrow U$, a function mapping each session s to a single user. user(s) is constant within s.
- permissions: $R \rightarrow 2^P$, a function mapping a role to a set of assigned permissions.
- roles: $S \rightarrow 2^R$, a function mapping a session to a set of roles, and $roles(s) \subseteq \{r|(\exists r' \geq r)[(user(s), r') \in UA]\}$, which may change within session s, and session s has the permissions $\cup_{r \in roles(s)}\{p|(\exists r'' \leq r)[(p, r'') \in PA]\}$

Through the concept of group in Definition 1, we introduce the concept of default group role set (*DSet*).

Definition 2. The default role set of a group DSet: $G \rightarrow 2^R$ is a subset of R, and $\forall u \in U, r \in R, (u, g) \in UM \wedge r \in DSet(g) \rightarrow (u, r) \in GUA$.

With this approach, a user who is mapped to a group obtains all the roles in the *DSet* of the group automatically.

The procedure to determine the permissions of a user in GB-RBAC is described as follows. When a user logins a system or starts an application, a session is created and a subset of the assigned roles of the user is activated. The set of assigned roles of a user includes the user's directly assigned roles (through SUA), the roles in the *DSet* of the group that the user is registered, and the roles that are assigned by group-level administrators (through GUA). The user obtains all the permissions assigned to these roles through PA. Users can also change the activated roles in a session within his assigned roles. The session can be terminated by the user or by the system, e.g., because of a long idle duration. For simplicity in this paper we assume that in a single session a user cannot change his group membership.

In GB-RBAC, we propose two layers of roles through groups: system-level roles (SR) which work in context of the overall system and group-level roles (GR) which work in the context of groups. In this way, besides SR, a user can be assigned to GR if s/he *belongs to* some groups. The user assigned to SR and GR gets different scopes of permissions. In addition, *DSet* provided in GB-RBAC enables a new user-role assignment mechanism to reduce administrative tasks. In this way, a new member of a group can be assigned some default roles without administrators' involvement, and group administrators can assign other explicit roles to group members based on roles in *DSet*. Fig. 3 illustrates these two mechanisms of user-role assignment. The upper part indicates SUA, which assigns users to roles directly. The lower part indicates GUA, where users are mapped to groups first and then assigned to roles in the corresponding

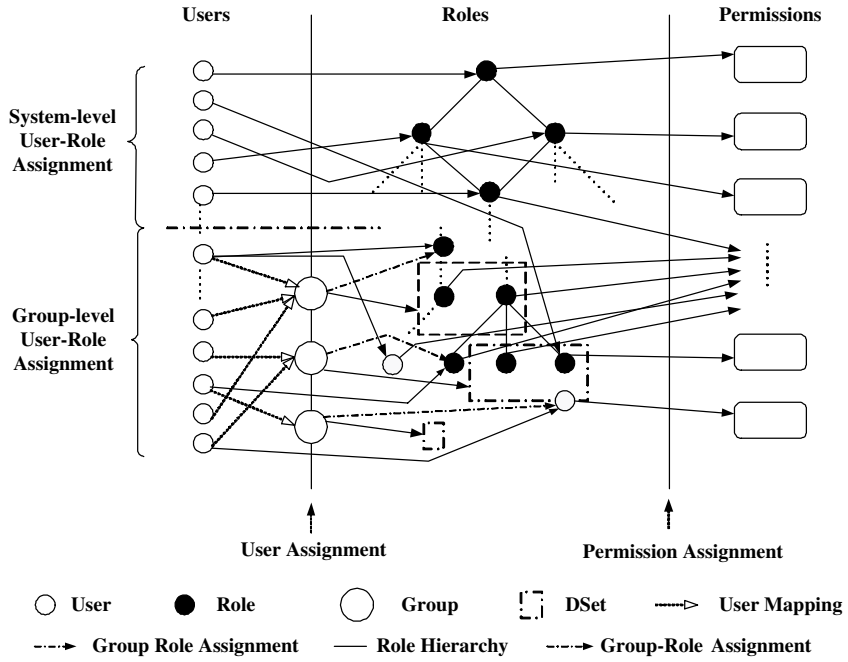

Fig. 3 – Two-level user-role assignments in GB-RBAC model.

groups. We discuss these user-role administrations in Section 4.

From the formal description of the model, we can see that the use of roles in *DSet* of a group does not take effect when the roles in the set have no permission assigned. In addition, *DSet* or a group can be changed by system or group administrators, thus affects the overall user-role assignment in GB-RBAC. The detailed administrative model is described in Section 4.

A GB-RBAC model can also have constraints defined on many aspects shown in Fig. 2. Besides the constraints on SUA, PA, RH, and sessions which are similar to those in RBAC96, GB-RBAC introduces new constraints on UM, GA, and GUA. This paper does not cover detailed specifications of constraints.

4. GB-RBAC administrative model

This section first gives an overview of user-role administration in GB-RBAC, then describes the formal model, and then discusses its advantages over traditional administrative models.

4.1. *Overview*

The success of an access control system is heavily dependent upon its administration, especially when the number of users and roles are on a scale of thousands. Managing access control components and their inter-relationships is an important and formidable task. Compared with previous RBAC models, our model introduces extra administration tasks, such as group-role assignment and user-group mapping. Two-level administrative models referred as system-level and group-level administrative model, respectively, are proposed to manage the relations defined in GB-RBAC. The administrative model of user-group mapping (UM) is to classify the users into different groups and it is the duty of the system-level administrative model. The group-role assignment (GA) is similar to the user-role assignment to some extent, which is in charged by system-level administrators. Besides these, *DSet* of a group is another administrative task that can affect the permission propagation in the model. The management of *DSet* of a group can be implemented in both administration levels. However, we consider it in group-level administration because one of the motivation of the model is to provide group-level autonomy administration, such that a group administrator has the permission to assign users to the roles in the GA relation.

For these two administration levels, two types of administrative roles are defined in the administrative model, called system-level administrative roles (SAR) and group-level administrative roles (GAR). These administrative roles can also form role hierarchies, respectively, similar to that of the regular roles in GB-RBAC. For simplicity we assume that $SAR \cap GAR = \emptyset$. A user of a system administrative role (or simply, a system administrator) can assign a user to a group (through UM), but a user of a group administrative role (or simply, a group administrator) can determine which role the user can be assigned to. In this way, a type of separation of duty in different levels of administration is provided. In addition, UM can be managed by an administrative model simpler than GA, and it does not add much complexity into the administrative model of GB-RBAC.

As Fig. 4 shows, a two-level administrative model referred as system-level and group-level administrations, respectively, is proposed to address all kinds of components defined in the

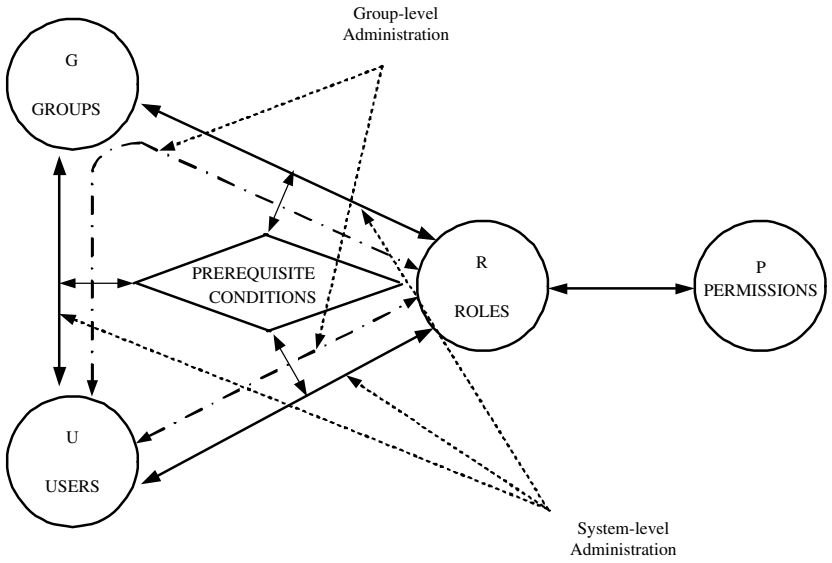

Fig. 4 – GB-RBAC administrative model.

GB-RBAC. Our administrative model discussed in this paper focuses on user-role assignment, which addresses the management the following components in GB-RBAC: GA, UM, SUA, GUA, and *DSet*. Specifically, system-level administrative model has three types of controls enforced on GA, UM, SUA, respectively, while group-level administrative model has two types of controls enforced on GUA and *DSet*, respectively. Note that we do not address the administration of UA since it can be implemented by the administration of SUA and GUA through Definition 1. Also note that as the control on *DSet* of a group defines the default role set of the group members, it implicitly manages the user-role assignment. For example, in a temporal group telephone conference, a role with common permission of connecting the virtual conference room is defined in the *DSet*, which is assigned to all users in the group by default. As a group's *DSet* is very application-specific, we do not explicitly define the rules to manage *DSet* in this paper.

Different layers of administrations provide an autonomy mechanism such that local administrators of a group can assign a member of the group to different roles if some assignment conditions are satisfied (introduced shortly). It is another flexibility to administrate UA besides the mechanism provided by *DSet* to deduce administrative tasks in first level administrative model.

Different level administrative models have different responsibilities: administration of GB-RBAC components with centralized control over users is in the system-level administrative model, and administration in the group view is in the group-level administrative model. Fig. 4 shows the scope of these two-level administrations. The system level model operates in the system scope to assign/revoke system-level roles to/from users, assign/revoke group-level roles to/from groups, and map/unmap user to/from groups, while the group level model operates in the group scope to assign/revoke roles to/from users, including defining roles in *DSet*. For sake of simplicity we do not consider administrations of permissions and role hierarchy in this paper.

4.2. *GB-RBAC grant model*

The grant model defines policies or rules authorizing administrators to assign users to roles and groups. Before explaining the details of these rules, the notions of prerequisite conditions in different types of assignments are introduced.

4.2.1. *User and group prerequisite conditions*

Definition 3. A user prerequisite condition is a boolean expression using the usual $\wedge$ and $\vee$ operators on terms of the form x and $\bar{x}$, where x is a regular role (i.e., $x \in R$) or a group (i.e., $x \in G$). A prerequisite condition is evaluated for a user u by interpreting x to be true if any of the follows is true:

- if $x \in R$, $\exists x' \geq x$, $(u, x') \in UA$;
- if $x \in G$, $(u, x) \in UM$.

and interpreting $\bar{x}$ to be true if any of the follows is true:

- if $x \in R$, $\forall x' \geq x$, $(u, x') \notin UA$;
- if $x \in G$, $(u, x) \notin UM$.

For a given set of roles R and G, let CR_u denote all possible user prerequisite conditions that can be formed.

A user prerequisite condition tests a user's memberships/nonmemberships of roles and groups. As the membership of a role tests both SUA and GUA, the prerequisite condition defined above is at least as expressive as that in URA97 (Sandhu et al., 1999).

Definition 4. A group prerequisite condition is a boolean expression using the usual $\wedge$ and $\vee$ operators on terms of the form x and $\bar{x}$, where x is a regular role (i.e., $x \in R$). A prerequisite condition is evaluated for a group g by interpreting x to

be true if $\exists x' \geq x$, $(g, x') \in GA$, and interpreting $\overline{x}$ to be true if $\forall x' \geq x$, $(g, x') \notin GA$. For a given set of roles R, let CR_g denote all possible group prerequisite conditions that can be formed.

A group prerequisite condition checks GA relation to test the memberships/nonmemberships of a group, which is used in the administration of group-role assignment.

4.2.2. *User-role assignment*

Authorizations on user-role assignment of GB-RBAC are controlled by a set of rules.

Definition 5. In system-level administrative grant model,

- user-role assignment in SUA is controlled by means of the relation can_assign_SUA $\subseteq SAR \times CR_u \times 2^R$.
- user-group mapping in UM is controlled by means of the relation can_assign_UM $\subseteq SAR \times CR_u \times 2^G$.
- group-role assignment in GA is controlled by means of the relation can_assign_GA $\subseteq SAR \times CR_g \times 2^R$.

Definition 6. In group-level administrative grant model,

- user-role assignment in GUA is controlled by the relation can_assign_GUA $\subseteq GAR \times CR_u \times 2^R$.

Specifically, a relation in above two definitions has three parameters (x, y, {z}), which means that a member of x can assign a user/group to be a member of a role in role range {z} if the user/group satisfies the corresponding prerequisite condition y. Specifically, a relation *can_assign_SUA*(x, y, {z}) or *can_assign_GUA*(x, y, {z}) means that a member of the system or group administrative role x (or an administrative role senior to x) can assign a user to be a member of a role in role range {z} if the user satisfies the prerequisite condition y; a relation *can_assign_UM*(x, y, {z}) means that a member of the system administrative role x (or an administrative role senior to x) can assign a user to be a member of a group in {z} if the user satisfies the prerequisite conditions y; and a relation *can_assign_GA*(x, y, {z}) means that a member of the system administrative role x (or an administrative role senior to x) can assign a group to a role in role range {z} if the group satisfies the prerequisite conditions y. Note that in a GB-RBAC model, users (e.g., user accounts), roles and permissions are created by the system administrators,[3] and group administrators can manage their relations in the group level.

As illustrated in the lower part of Fig. 3, the blank circles in roles column denote administrative roles and the solid circles in roles column denote common roles. If a user in a group is assigned to an administrative role, s/he can assign roles to the members of the group after successful testing of the prerequisite condition. In this way, members of the group can be assigned proper roles through GUA. In addition, the upper part of Fig. 3 illustrates another user-role assignment mechanism that a user also can be assigned roles through SUA.

[3] Note that the administrative model introduced in this paper does not include corresponding rules to create users, roles, and permissions, as well as role hierarchy administrations.

Table 1 – Administration control rules.

Type	Admin. Role	Rrereq. Condition	Group/Role Range
can_assign_SUA	E-SSO	resAA	resAD
can_assign_UM	E-SSO	resAA	{@PRO1}
can_assign_GUA	PM	@PRO1 $\wedge \overline{QE1}$	{PE1}

As an example, consider a set of administration rules defined in the organization as Table 1 shows. We put an "@" in front of the group names to distinguish with role names. A group (PRO1) is created to develop a group level administrative domain. Roles are created as shown in Fig. 5. The part above the dashed line presents system-level roles, while the part below the dashed line presents group-level roles. These two levels of roles both contain two types of roles: normal roles such as resAA (resource A access) in the system-level roles and PL1 in the group-level roles, and administrative roles such as S-SSO in the system-level roles and GD in the group-level roles. Role hierarchies also exist among these roles. For example, there is a role hierarchy between the two system-level roles: a junior-most role resAA and a senior-most role resAO (resource A owner). Between them, there are two other incomparable roles, resAD (resource A dissemination) and resAM (resource A modification).

System-level role assignment is similar to that in URA97 (Sandhu et al., 1999). For example, if Alice is a member of E-SSO and Bob is a member of resAA, she can assign Bob to role resAD, according to the first rule in Table 1. At the same time, according to the second rule, Alice can assign Bob to group PRO1. Now we consider administration in the group level. We assume Carol is a member of PM and Bob is a member of group PRO1. Carol can assign Bob to PE1 if Bob is not a member of QE1, according to *can_assign_GUA*(PM,@PRO1 $\wedge \overline{QE1}$, {PE1}).

Note that in GB-RBAC, mapping roles to groups (that is, to define GA) is typically very application-specific. Our administrative model does not include rules for this purpose. For example, in Fig. 5 we assume that the roles in PRO1 are pre-defined.

An assumption in our administrative model is that a system administrator is trusted not to assign roles to conflicting groups. For example, in above case, as role range [ER1, PL1] has been assigned to group PRO1, Alice and other administrators should not assign any of the role in this range to other group[4] (say PRO2), otherwise a user from PRO2 can have permissions in PRO1, e.g., to read/write sensitive data, which is not allowed in general. This assumption is also used in traditional RBAC administrative models, e.g., administrators are trusted not to assign two conflict roles to a single user. Constraints can be defined to restrict the permissions of administrators, which are not covered in this paper.

4.3. *GB-RBAC revocation model*

The revocation rules in GB-RBAC are controlled by *can_revoke* relations.

[4] In this paper we do not consider group hierarchy in the model. If a group belongs to another group, then a role can be assigned to both of them.

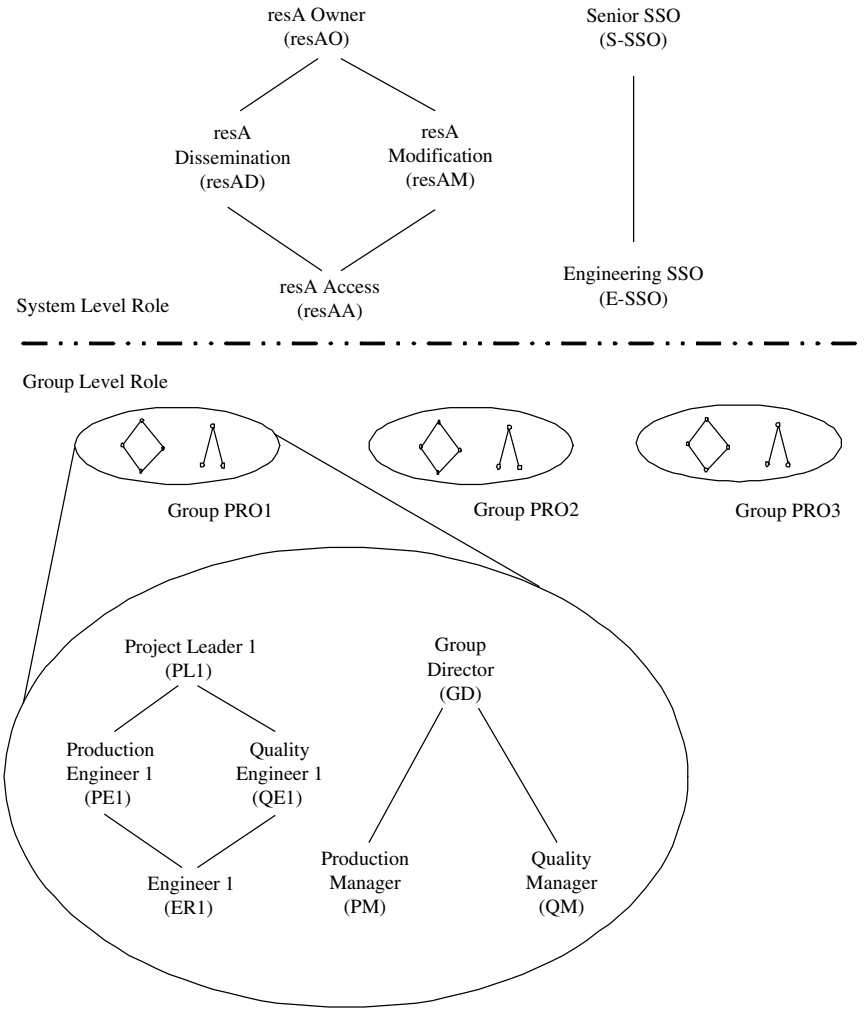

Fig. 5 – Example: different levels of roles in GB-RBAC.

Definition 7. In system-level administrative revocation model,

- user-role revocation in SUA is controlled by means of the relation can_revoke_SUA $\subseteq$ SAR $\times 2^{R}$.
- user-group unmapping in UM is controlled by means of the relation can_revoke_UM $\subseteq$ SAR $\times 2^{G}$.
- group-role revocation in GA is controlled by means of the relation can_revoke_GA $\subseteq$ SAR $\times 2^{R}$.

Definition 8. In group-level administrative revocation model,

- user-role revocation in GUA is controlled according to the relation can_revoke_GUA $\subseteq$ GAR $\times 2^{R}$.

Specifically, a relation *can_revoke*(*x*, {*z*}) specifies that an administrative member of role *x* can revoke a user or a group from a role or group in {*z*}. Similar to that in URA97 (Sandhu et al., 1999), the user-role revocation model (in both system-level or group-level) of GB-RBAC has two types of operations: weak revocation and strong revocation. Specifically, for $(u, r) \in UA$, a weak revocation on it has no effect if *u* is not explicitly assigned to *r*, that is, there exists $r' \leq r$ such that $(u, r') \in UA$; while a strong revocation on (*u*, *r*) tries to do weak revocation on (*u*, *r*′) for every $r' \geq r$, where (*u*, *r*′) is an explicit assignment.

Let us consider a set of revocation rules in Table 2 and interpret it in the context of Fig. 5. Let Alice be a member of E-SSO, and Bob a member of resAD. Consider Alice try to revoke membership of Bob from role resAA: with weak revocation, this has no effect since Bob is still a member of resAD; while with the strong revocation, Alice can revoke Bob from resAA according to the first rule in Table 2. Now we consider revocations in group level. Let Alice be a member of E-SSO, and Bob a member of group PRO1 and role PE1. With rule *can_revoke_UM*(E-SSO,@PRO1), Alice is authorized to revoke the membership of Bob from group PRO1. With weak revocation, this has no effect since Bob is still a member of PE1. But this revocation prevents Bob from being assigned to other roles in

Table 2 – Revocation control rules.

Type	Admin. Role	Group/Role Range
can_revoke_SUA	E-SSO	[resAA, resAD]
can_revoke_UM	E-SSO	{@PRO1}
can_revoke_GUA	PM	(ER1, PL1)

the group, such as to PL1 by GD (refer to Table 1). A strong revocation from Bob's membership of PRO1 revokes Bob's all memberships of roles in PRO1. Within group PRO1, rule *can_revoke_GUA*(PM, (ER1,PL1)) indicates that Carol who is a member of PM can revoke Bob from PE1.

4.4. Discussion

GB-RBAC and its administrative model are the fundamental work for our secure collaboration scheme shown in next section. We summary the main advantages of our approach by comparing it with ARBAC97 as follows.

(i) *Simplified User-role Assignment for System Administrators* In our model, an administrator only needs to assign a user to a group and specify the role range of a group. After that, group administrators take charge of the user-role assignment in this role range. This significantly simplifies the management task by delegating administrative permissions from centralized system-level administrators to decentralized group-level administrators, especially for dynamic and ad-hoc group-based applications. In contrast, in ARBAC97, when a new project or a new department is introduced in the system, a set of roles are created and corresponding rules have to be defined to manage it. But in our administrative model, a system administrator only needs to create a group and user-group mapping relation, the other assignment can be managed by the group-level administrators locally.

(ii) *Flexible Administration for Dynamic User-role Assignment* Group-level administration can easily support dynamic user participation in group-based applications. For example, consider a voice-over-IP (VoIP) conference is to be held within group PRO1 in Fig. 5. Based on the permissions illustrated in Table 3, the members of PL1, PE1, QE1 and ER1 can join this conference, and the members of PL1, PE1 and QE1 can speak in the conference, and only the members of PL1 can host this conference. During a conference users can join and leave. With purely system-level administration, it is tedious to do the dynamic user-role assignment, while it is very efficient with group-level administration. For example, according to the group-level administrative rule in Table 2, any member of PM can administrate user-role assignment in this group.

(iii) *Fine-grained User-role Assignment* By enabling GUA, our model supports user-role assignment in the group level. Typically, a group administrator has more contextual information about the permissions and sensitive operations in the group and the users' skills, thus user-role assignment in this level provides fine-grained user management and less risk of authorizations (Nissanke and Khayat, 2004).

(iv) *Tunable Group-level Administrations* A system-level administrator can change the role assignment of a group, and thus change the roles that a group administrator can assign users to. For example, with the rules in Tables 1 and 2, Alice can change PRO1's memberships of roles from [ER1, PL1] to [ER1, PL1) by revoking (PRO1, PL1) from GA. This greatly provides flexible and controlled group-level administrative permissions.

Table 3 – Example of role assignment.

Role	Permissions	Role	Permissions
PL1	conf1_host	PL2	conf2_host
PE1	conf1_speak (P1) prog1_upload (P2)	PE2	conf2_speak prog2_upload
QE1	conf1_speak prog1_report	QE2	conf2_speak (P3) prog2_report (P4)
ER1	conf1_join	ER2	conf2_join

5. Supporting Ad-hoc collaborations with GB-RBAC

This section first identifies the generic access control requirements for ad-hoc collaborations, and then presents our solution with GB-RBAC and proposes algorithms to build group-level roles and their permission assignments.

5.1. Ad-hoc collaboration scheme

We first identify two important features of ad-hoc collaborations which determine access control requirements. In this paper, a group identifies an autonomous control domain. A collaboration scheme should enable management autonomy in individual groups and information exchangeability between groups.

Previous work result in many violations/conflicts (Shafiq et al., 2005; Kapadia et al., 2000) when a collaboration between different groups happens. Most of previous work implement collaborative through direct role mapping relations between groups, where violations or problems happen such as user-specific SoD violation, role-specific SoD violation, role-assignment violation (Shafiq et al., 2005), and role promotion (Kapadia et al., 2000). In our work, we propose the concept of virtual group. Roles involved in a collaboration are *exported* into a virtual group from their original (source) groups. In this way, most of violations/problems are solved such as SoD violations mentioned above, and constraints such as induced SoD (Shafiq et al., 2005) are eliminated at user-role assignment stage.

In order to support ad-hoc collaboration with GB-RBAC, we propose a special group: virtual group (VG). A VG has the similar features as common groups except that it only contains group-level components (roles and permissions) exported from collaborating groups. Fig. 6 illustrates application of a VG.

For a collaboration between several collaborative groups, the building procedure is briefly implemented as following steps: 1) A collaboration request is sent to collaborative groups

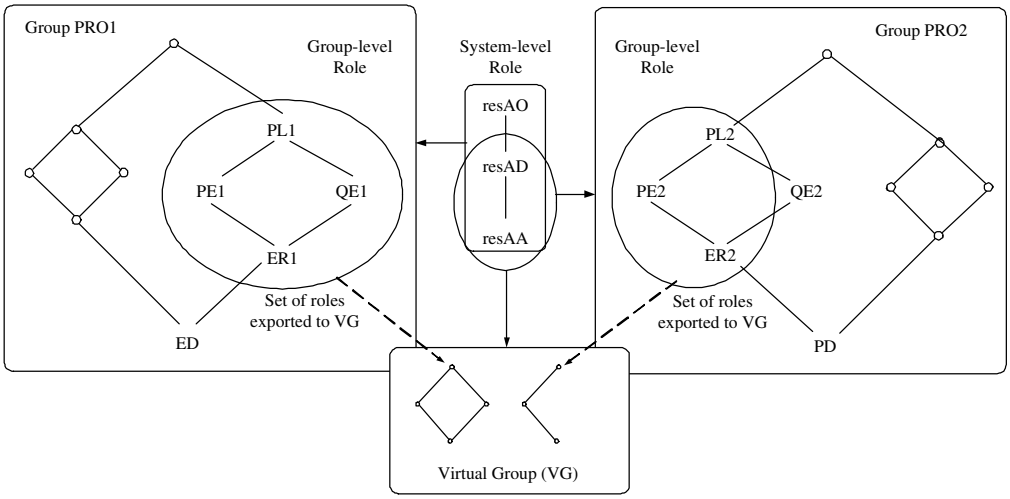

Fig. 6 – Collaboration between groups with GB-RBAC.

by an administrator of an initial group (called the VG founder), and the response is sent back to the founder. 2) Upon request, administrators of all collaborative groups build VG using ColGrant Algorithm. In ColGrant algorithm, roles and permissions from collaborative groups are exported into VG based on collaboration policies. 3) VG administrators are elected from the collaborative group administrators. For simplicity, in this paper we assume that all the collaborative group administrators are the VG's administrators. 4) User-role assignment can be performed by the VG administrators by the user-role administrative model in Section 4. 5) All the members of VG can start the collaborative work, and some collaboration modification can be realized using ColUpdate Algorithm. In ColUpdate algorithm, roles and permissions in VG are updated to meet requirements of changed collaboration polices. 6) The collaborative work finishes, and the administrators of collaborative groups who leave the VG last destroy the VG with ColRevo algorithm. In ColRevo algorithm, roles and permissions are revoked from VG.

In Fig. 6, PRO1 exports {ER1, PE1, QE1, PL1} to VG, and PRO2 exports {ER2, PE2, PL2} to VG. In this way, VG *contains* roles {ER1, ER2, PE1, QE1, PE2, PL1, PL2} and corresponding permissions. In this way, all members assigned to roles in VG can act as appropriate roles in collaborative work, no matter where the roles come from originally.

Before defining the three algorithms for the basic operations in a collaboration, we give the definition of role conflict which may exist when exporting roles in different groups to a VG, and the definition of role naming mechanism which is used to solve conflictions.

5.2. *Role conflicts*

In our collaboration scheme, we consider two type of conflicts: name conflicts and role conflicts. A name conflict happens when exporting a role into a VG which has the same name of another role in the VG. Role conflict is an advanced concept and has the following definition.

Definition 9. Role r_j conflicts with r_i in a virtual group VG if $\exists$p1, p2, p3, p4 $\in$ P, (p1, p2) $\subseteq$ permissions(r_i) $\wedge$ (p3, p4) $\subseteq$ permissions(r_j) $\wedge$ linked(r_i, VG_y) $\wedge$ $\neg$linked(r_j, VG) $\wedge$ $\diamond$(p1, p3) $\wedge$ $\neg\diamond$(p2, p4), where linked(r, VG) is a predicate that tests whether r has been exported into VG, and $\diamond$ denotes that two permissions can be obtained by a user through role r_i and r_j simultaneously.

A role conflict between r_i (in a VG) and r_j (not in VG) happens if there exist two permissions from r_i and r_j that can be obtained by a user while there are other two permissions that cannot be obtained by the user simultaneously, e.g., according to pre-defined SoD constraints. We say that r_j is a *conflicting* role for VG. For example, as illustrated in Fig. 7(c), P1 and P2 are permissions contained in role QE2, P3 and P4 are contained in role PE1 (The details of permission can be found in Table 3). Although P1(conf1_speak) and P3(conf2_speak) can be simultaneously achieved by a user (that is, the user can speak both at conference 1 and conference 2), P2 and P4 must be exclusively achieved by the user because the user should not simultaneously have the permissions to perform the program1 upload operation and program2 report operation, according to the organization's policy. Based on Definition 9, there exists role conflict between these roles.

If a role conflict exists, we need split role r_j into two parts, e.g., by creating two roles and assigning P2 and P4 of QE2 to these two roles, respectively. In order to simplify role collaboration scheme, a naming mechanism is defined as follows.

Definition 10. When exporting the components of a collaborative group into a virtual group,

- if a name conflict exists, the new name of the exporting role is its original name plus the collaborative group name;
- if a role conflict exists, the new name of the conflicting role is its original name plus the serial number of every part after dividing the role.

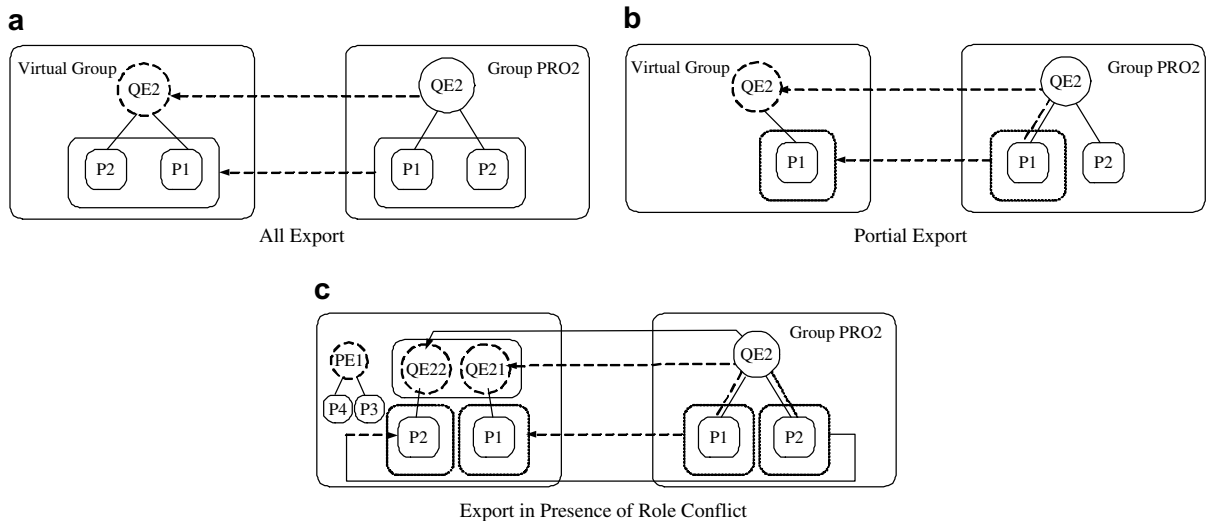

Fig. 7 – Role export scheme in GB-RBAC.

For example, if a name conflict exists when QE1 from PRO1 is exported into VG, we name the role QE1PRO1. If a role conflict exists, we divide a conflicting role into two parts. For example, in Fig. 7(c), QE2 conflicts with PE1 which is already export to VG. Now we split QE2 and the name of every part of QE2 is named as QE21 and QE22.

In general, we consider three cases when roles and permissions are exported:

(1) Roles which can be directly exported into VG;
(2) Roles which can be partly exported into VG; That is, the subset of permissions obtained by the roles can be exported.
(3) Roles which should be wholly exported into VG. However, some of them conflict with existing roles in VG, and the set of permissions obtained by these roles should be exported individually.

Fig. 7 illustrates the scenario where QE2 of PRO2 is exported to VG in different cases. In the first case, we simply export all permissions of QE2 to VG, and we can also directly export QE2. In the second case, only subset of permissions of QE2 can be exported to VG. Specifically, we split the permissions of QE2, and export the part with permission P1 to VG. In the third case, QE2 and PE1 conflict in VG. We split the permissions of QE2 into subsets P1 and P2, create two new roles QE21 and QE22, and assigned with P1 and P2, respectively. After that, QE21 and QE22 are be exported to VG, respectively, and P1 and P2 can also be exported to VG.

All users and permissions in a virtual group do not contain any genuine information. Actually, they are only link information which denotes which component comes from which group. However, a role in a virtual group is assigned with permissions of its linked role. The names of the components in virtual group directly use or derive from those of the original components of collaborative groups.

5.3. *Operations for collaboration*

With defined mechanisms to resolve role collaborations, we present the three algorithms which are used to build a virtual group, update the components of a virtual group, and destroy a virtual group, respectively. After a virtual group is established, assigning users to roles in the group can be performed by the group administrators following the user-role administrative model presented in Section 4. So in this section we focus on the details of exportation of roles and permissions.

ColGrant algorithm shown in Fig. 8 describes the main steps to build a virtual group among collaborative groups. In the process of collaboration building, one of the administrators[5] from the VG founder group creates a virtual group name with the necessary parameters, including the names of other collaborative groups. The algorithms starts by exporting the founder group's roles and permissions to the VG. Two cases are considered here: If all permissions included in the group roles need to be exported, the administrator directly exports the roles and the corresponding permissions by using *role-link* (r, VG_y) function; If only a subset of the permissions need to be exported, the algorithm first creates a role link in the context of the virtual group using *createrole* function, and then fetches permissions included in the roles of the group. Through *insert(r, p)* function, the links of the corresponding permissions are added into *r* if the test succeeds in the exportable permissions using *export-permission* function. If all the permissions are inserted into the new role link, the link is attached to VG through *link-role* function. After that, the algorithm achieves the updated *DSet* and assigns users to the virtual group. This process makes sure that the virtual group is created and the components of the initial group is exported to the virtual group. Following the similar process, the

[5] The capability to initialize a collaboration is a group-level administrative permission, which is not specified in this paper.

```
ColGrant Algorithm
1)   DSet_tmp ← G_x.DSet
2)   if VG_y = ∅
3)       VG_y ← creategroup()
4)       VG_y = createVG();
5)       for each role r_i ∈ G_x.R_set
6)           if all-export(permissions(r_i))
7)               role-link(r_i,VG_y)
8)           else if part-export(permissions(r_i))
9)               r_new ← createrole()
10)              for each p_i ∈ permissions (r)
11)                  if export-permissions(p_i)
12)                      insert(p_i,r_new)
13)                      link-role(r_new,VG_y)
14)              if r_i ∈G_x.DSet
15)                  DSet_tmp ←DSet_tmp∪r_new - r_i
16)      VG_y.DSet ←VG_y.DSet ∪ DSet_tmp
17)  else
18)      for each role r_i ∈G_x.R_set
19)          if name-conflict(r_i,VG_y)
20)              r_i ← name-change(r_i)
21)-30)  similar with step 6)-15), we do not repeat it again
31)          else if permission-conflict(r_i, VG_y)
32)              permissions_con ← conflict-permissions(r_i, VG_y)
33)              r_new ← createrole()
34)              if export-permissions(permissions_con)
35)                  insert(permissions_con,r_new)
36)                  insert(permissions(r_i)-permissions_con, r_res)
37)                  link-role(r_new,VG_y)
38)                  link-role(r_con, VG_y)
39)              if r_i ∈ G_x.DSet
40)                  DSet_tmp ← DSet_tmp∪ r_new∪ r_res - r_i
41)      VG_y.DSet ← VG_y.DSet ∪ DSet_tmp
```

Fig. 8 – ColGrant algorithm.

administrators of other participant group can export necessary roles and permissions to the virtual group. In each step we check whether roles in the original group have identical names with those in the virtual group. If there is any role name conflict, we change the role name using *name-change* function and use the modified name as the name of role link. Now we consider the three cases aforementioned and export roles in a sound way. We have already mentioned the first two cases in the first step. In the third case, we create two new role links using *createrole* function and insert the links of the permissions into the corresponding role links. After attaching the two role links to VG, we evaluate *DSet* of the virtual group, which is the union of *DSet*s of all the collaborative groups.

Now we consider some examples about ColGrant algorithm with Fig. 7. In the first state, we assume that the administrator of PRO1 creates the virtual group (VG), and exports all the roles of ER1, PE1, QE1 and PL1 and corresponding permissions to VG. Because we achieve the permissions of roles through *permissions(r)*, these exporting procedures are implemented by the *role-link* function. We assume that *DSet* in PRO1 is {ER1}, and we unite this set into *DSet* of VG. So *DSet* of VG is {ER1}. In the second stage, PRO2 starts to join VG using the algorithm. Because there exist no name conflict and role conflict with the roles in current VG, we directly export the roles {ER2, PE2, PL2} and corresponding permissions to VG. We assume that *DSet* of PRO2 is {ER2, PE2}, and we also unite this set into *DSet* of VG and the value of *DSet* is {ER1, ER2, PE2}. With these steps, a simple process of virtual group building is finished. The administrators of PRO1 and PRO2 become the administrators of the virtual group, and these administrators can assign roles to users through *can_assign_GUA* in group level administrative model discussed in Section 4. In this way, the users in the virtual group can be authorized with permissions and start the collaborative work with each other.

Algorithms ColUpdate (see Fig. 9) and ColRevo (see Fig. 10) are used to update the components of a virtual group and delete a virtual group, respectively. Because the process to add/delete a component into/from a virtual group is similar to that in ColGrant, we do not present these issues in the ColUpdate algorithm. In ColUpdate algorithm, we use a flag to distinguish the different cases of role export. If the permissions of a role are updated, e.g., in its original group, we unlink the role and re-export the updated role in a virtual group. In the ColRevo algorithm, we also need a flag to distinguish the three different cases. If a role is directly exported into a virtual group, we delete the role link. However, if the role is split into

```
ColUpdate Algorithm
1)   if action = add
2)       similar with ColGrant Algorithm
3)   else if action = del
4)       similar with ColRevo Algorithm
5)   else if action = mod
6)       Initial Flag ← 0;
7)       if G_x.R_set ≠ ∅
8)       for each role r_i ∈G_x.R_set
9)           r_t ← name-change(r_i)
13)          if FindRole(r_t) = false
14)              r_t ← r_i
15)          if FindRole(r_t) = false
16)              Flag ← 1
17)          else
18)              r_new, r_res ← name-tranform(r_t)
19)              Flag ← 2
20)          if permission-conflict(r_t, VG_y)
21)              permissions_con ← conflict-permissions(r_t, VG_y)
22)              if Flag = 2
23)                  permission-update(r_new,permissions_con)
24)                  permission-update(r_res,
                       permissions(r_i)-permissions_con)
25)              else
26)                  role-unlink(r_t,VG_y)
27)                  r_new ← createrole()
28)                  insert(permissions_con,r_new)
29)                  insert(permissions(r_i)- permissions_con, r_res)
30)                  if (export-permissions(permissions_con)
31)                    link-role(r_new,VG_y)
32)                    link-role(r_con, VG_y)
33)          else
34)              if Flag = 2
35)                  role-unlink(r_new,VG_y)
36)                  role-unlink(r_res,VG_y)
37)                  role-link(r_t,VG_y)
38)              else
39)                  permission-update(r_t,permissions(r))
40)          if r ∈G_x.DSet
41)              update the role name in G_x.DSet
42)      update VG_y.DSet using G_x.DSet
```

Fig. 9 – ColUpdate algorithm.

ColRevo Algorithm

1) $G_x.R_{set} \neq \emptyset$
2) **for** each role $r_i \in R_{set}$
3) $r_{tmp} \leftarrow$ name-change(r)
7) **if** FindRole(r_i) = false
8) $r_{tmp} \leftarrow r_i$
9) **if** FindRole(r_t) = false
10) Flag $\leftarrow 1$
11) **else if**
12) r_{new}, $r_{res} \leftarrow$ name-trnaform(r)
13) Flag $\leftarrow 2$
14) **if** Flag = 2
15) role-unlink(r_{new},VG_y)
16) role-unlink(r_{res},VG_y)
17) **else**
18) role-unlink(r_{tmp},VG_y)
19) **if** users(VG_y) = $\emptyset \wedge$ roles=(VG_y) = $\emptyset$
20) deleteVG()

Fig. 10 – ColRevo algorithm.

two roles when exported to the virtual group, we should transform the role name and delete the role links. If there exists no component in a virtual group, it can be destroyed. Again, revoking users from the roles in a virtual group is ignored here.

Note that a role link in a VG should be deleted and the role should be re-exported when the role export case are different in the different stages in the process of ColUpdate. For example, we should consider the case that the permissions of a role are entirely exported into VG and role conflicts happen in ColUpdate.

6. Prototype implementation and evaluation

To show the feasibility and performance of our approach, we implement a prototype system by enhancing the extensible access control markup language (XACML) with GB-RBAC model. This Section first gives an overview of our prototype and then presents some results of performance study.

6.1. *Policy specification*

Our prototype uses the extensible access control markup language (XACML) to specify GB-RBAC policies. XACML is an open-standard format to specify access control policies, and expected to be widely used with the properties of interpretability and extensibility. Using the Sun's XACML library, the Policy Decision Point (PDP) module interprets XACML policies and makes access decisions.

In our prototype, both permission-role and user-role policies are located in an authorization service platform. Fig. 11 shows the skeleton of two sample policies of them, respectively. In these two policies, PRO1_host is a role name in PRO1. The first policy states that this role has the permission to host in the group PRO1 only when no user acts as host of PRO1 or the user has the same group (PRO1) id as that of users who are authorized as PRO1_host at the same time, which is specified by Domain 1. The second policy states that the user whose

```
<PolicySet PolicySetId="Domain1:Role:Permission">
  ...
  <Subjects>...urn:mynamespace:role:PRO1\_host...</subjects>
  <Resources> ... invite_speaker ... </Resources>
  <Actions> ... PRO1 ... </Actions>
  ...
  <Condition>
    <EnvironmentAttributeDesigner AttributeID="group-id"/>
    <AttributeVaule>equal</AttributeValue>
  </Condition>
  ...
</PolicySet>


<PolicySet PolicySetId="PRO1:User:Role">
  ...
  <Subjects>
    <SubjectMatch>Alice</SubjectMatch>
    <SubjectMatch>urn:mynamespace:group:PRO1</SubjectMatch>
  </Subjects>
  <Resources>...urn:mynamespace:role:PRO1\_host...</Resources>
  <Actions> ... membership ... </Action>
  ...
    ...
  <Condition>
    ...
      <EnvironmentAttributeDesigner AttributeID="time"/>
      <AttributeVaule>9AM</AttributeValue>
    ...
      <EnvironmentAttributeDesigner AttributeID="time"/>
      <AttributeVaule>7PM</AttributeValue>
    ...
  </Condition>
  ...
<PolicySet>
```

Fig. 11 – Policies for permission-role and user-role assignments in prototype.

name is Alice within group RPO1 is assigned with role PRO1_host only during the hours 9AM to 7PM, which is specified by PRO1. The net effect of these two policies specifies the user whose name is Alice within PRO1 has permission to invite speaker in PRO1 on Domain 1 only during 9AM to 7PM if no user acts as host of PRO1 or the user has the same group (PRO1) id as that of users who are authorized as PRO1_host.

6.2. *Performance evaluation*

As a GB-RBAC decision is determined by verifying subject (requesting user) credentials, objects (resources) and actions, the performance of GB-RBAC policy process should be considered. Firstly, we evaluate the overhead introduced by decentralized administration compared to original XACML with RBAC profile. Secondly, we evaluate the performance when user authorization is performed based on the dynamic generated VG policies.

In our experiment, permission requests are generated in application service platforms and sent to authorization server which is in Java 1.4.2 and working in Windows XP machine with Pentium M 1.7 GHz and 512 MB memory. Fig. 12 shows the performance of per-authorization request with original XACML specified by the RBAC profile and enhance XACML with the GB-RBAC profile. The time of policy process with RBAC ranges from 0us to 16us, and the process time of per-authorization with enhanced XACML specified by the GB-RBAC profile ranges from 0us to 47us. The average policy process time for per-authorization with RBAC policies is about 7.92us and the average time with GB-RBAC is about 9.95us,

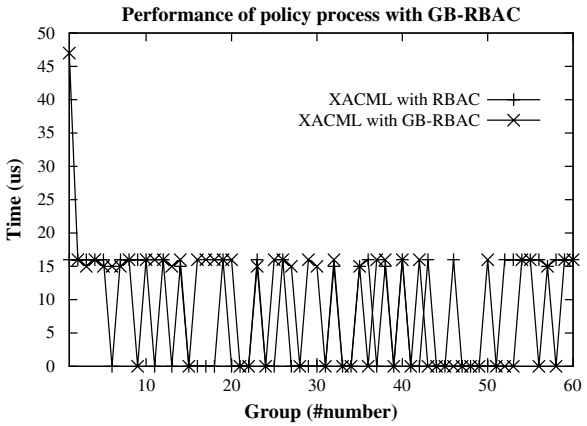

Fig. 12 – Performance of policy process with 60 group members.

which implies that our decentralized access control mechanism introduces about 25% extra overhead. As user authorization is a one-time operation when the authorization request is generated from application servers, this performance overhead introduced by decentralized RBAC administration is reasonable.

We further study the performance of ad-hoc collaborations. Fig. 13 illustrates the results of the performance of policy process without VG, with 1 VG and with 60 VGs, respectively. The authorization operation is conducted as follows: policy process without VG evaluates the policy based on static GB-RBAC policy defined in Fig. 11 and is to authorize concurrent requests for 60 group members from the specified group; policy process with 1 VG evaluates the policy which is dynamically generated for *VG1* based on the policy specified in Fig. 11 and is to authorize concurrent requests for 60 group members from *VG1*; policy process with 60 VGs evaluates the policy which is dynamically generated for 60 VGs named from *VG1* to *VG60* and is to authorize concurrent requests for 60 members from 60 different virtual groups. For simplicity but without loss of generality, we only generate same GB-RBAC rules for every group members in VG and the VG policies are only generated from a static group policy, which is triggered by the administrators of different source groups. As shown in Fig. 13, the process time with generation of 1 VG is about 20us and average process time for per user authorization is 10.13us, which only introduces less than 1% overhead compared to that without VG generation. The user authorization time with generation of 60 different VGs is about 34.36us. As VG policy generation is only invoked by the first VG users (the administrators of VGs), and it will not introduce overhead into authorization of group members. Moreover, as we mentioned above, in most real world applications, access control is only checked once during a type of continuous operations. For example, reading a batch of files from a directory only checks if the user has the permission to read the directory at the beginning. Therefore we believe that the authorization performance is acceptable for securing typical distributed systems.

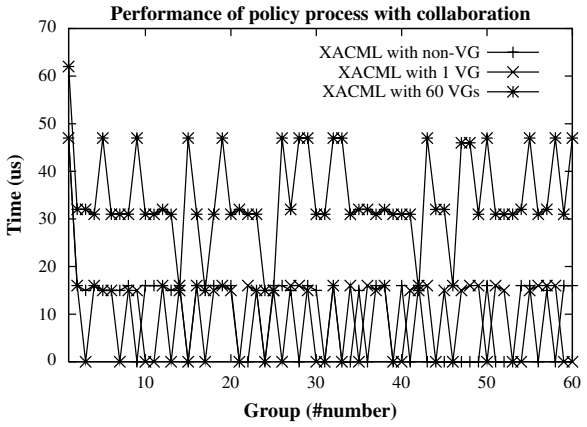

Fig. 13 – Performance of policy process with/without VG.

7. Conclusion and future work

In this paper, we present an advanced RBAC model called GB-RBAC for secure collaborations and its user-role administration. The main advantage of the model is convenience and flexibility for administration under large-scale environments. Our model does not need to be used in a highly central controlled environment, and provides two levels of administrative models for user-role assignment and reduces the complexity of the administration of RBAC systems. Moreover, user-role assignment in the group-level administrative model provides a flexible way to meet the requirements of group-level collaboration, i.e., users from different groups form a virtual group for communication. In addition, we propose an ad-hoc collaboration scheme in multi-group environment based on the GB-RBAC model. The scheme proposes a component of virtual group to enable secure collaborations between different groups. We present three algorithms to transform components from collaborative groups to virtual groups and allow them to access shared resources and information. In this way, our scheme provides a secure and easy solution to support ad-hoc collaborations. We implement a prototype with SunXACML and our experimental results demonstrate the efficiency and scalability of our authorization approach.

Several aspects need further study in our scheme. First of all, constraints for a virtual group need to be explored. We are going to provide an improved group-role assignment mechanism in which general constraints on group level can be supported. Especially, the proposed scheme needs to be extended to integrate advanced mechanisms and constraints to facilitate overall policy administration. Second, as the administrative model we proposed in this paper focuses on user-role administration, we are going to develop the group-level permission-role and role-role management for GB-RBAC. An integrated model that incorporates the permission pool (Oh et al., 2006) mechanism will be developed in the future. Based on the GB-RBAC model, small permission pools can be implemented in the group-level administrative model to provide a more flexible scheme for secure collaborations. Our

collaborative scheme requires collaborative parties have RBAC policies, which may not be always true in real application scenarios. However, as RBAC becomes standard (Ferraiolo et al., 2001; ANSI, 2004) and more and more organizations and systems adopt RBAC policy models, we believe our approach is applicable in many collaborations. We will fully explore some other issues in collaborations along with our model, such as policy conflicts and permission constraints between collaborative parties.

Acknowledgements

This research was supported in part by the China National Natural Science Foundation (NSFC) under grant No. 90604024, the National Basic Research Program of China (973 Program) under grant No. 2009CB320501, the National High Technology Research and Development Program of China (863 Program) under grant No. 2007AA01Z2A2, and the Key Project of Chinese Ministry of Education under grant No. 106012.

REFERENCES

ANSI. American national standard for information technology – role based access control, ANSI INCITS 359–2004, Feb. 2004.

Crampton J. Understanding and developing role-based administrative models. In: proceedings of 12th ACM conference on computer and communications security; 2005. p. 158–67.

Crampton J. Discretionary and mandatory access controls for role-based administration. In: proceedings of 20th annual IFIP WG 11.3 working conference on data and applications security; 2006. p. 194–208.

Crampton J, Loizou G. Administrative scope: a foundation for role-based administrative models. ACM Transactions on Information and Systems Security 2003;6(2):201–31.

Ferraiolo D, Sandhu R, Gavrila S, Kuhn D, Chandramouli R. Proposed NIST standard for role-based access control. ACM Transactions on Information and Systems Security 2001;4(3): 224–74.

Joshi J, Bhatti R, Bertino E, Ghafoor A. Access control language for multidomain environments. IEEE Internet Computing 2004: 40–50.

Kapadia A, AI-Muhtdai J, Campbell R, Mickunas D IRBAC 2000: secure interoperability using dynamic role translation. In: Technical Report: UIUCDCS-R-2000-2162; 2000.

Koch M, Mancini LV, Parisi-Presicce F. Administrative scope in the graph-based framework. In: proceeding of the 9th ACM symposium on access control models and technologies; 2004. p. 97–104.

Nissanke N, Khayat EJ. Risk based security analysis of permissions in rbac. In: proceedings of 2nd international workshop on information systems; 2004.

Nita-Rotaru C, Li N. A framework for role-based access control in group communication systems. In: proceedings of international workshop on security and parallel and distributed systems; 2004.

Nyanchama M, Osborn S. The role graph model and conflict of interest. ACM Transactions on Information and Systems Security 1999;2(1):3–33.

Oh S, Sandhu R, Zhang X. An effective role administration model using organization structure. ACM Transactions on Information and System Security 2006;9(2):113–37.

Osborn S, Guo Y. Modeling users in role-based access control. In: proceedings of 5th ACM workshop on role-based access control; 2000. p. 31–8.

Osborn S, Sandhu R, Munawer Q. Configuring role-based access control policies. ACM Transactions on Information and Systems Security 2000;3(2):85–106.

Park J, Sandhu R, Ahn GJ. Role-based access control on the web. ACM Transactions on Information and Systems Security 2001; 4(1):37–71.

Piromruen S, Joshi J. An RBAC framework for time constrained secure interoperation in multi-domain environments 2005:36–48.

Core and hierarchical role based access control (RBAC) profile of XACML v2.0, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf.

Sandhu R. Role versus group. In: proceeding of 1st ACM workshop on role-based access control; 1995. p. 1–12.

Sandhu R, Coyne E, Reinstein H, Youman C. Role-based access control model. IEEE Computer 1996;29(2):38–47.

Sandhu R, Bhamidipati V, Munawer Q. The ARBAC97 model for role-based administration of role. ACM Transactions on Information and Systems Security 1999;2(1):105–35.

Shafiq B, Joshi J, Bertino E, Ghafoor A. Secure interoperation in a multidomain environment employing RBAC poilcies. IEEE Transactions on Knowledge and Date Engineering 2005;17(11): 1557–77.

Sun's XACML, http://sunxacml.sourceforge.net/.

Tolone W, Ahn G, Pai T, Hong S. Access control in collaborative systems. ACM Computing Surveys 2005;37(1):29–41.

OASIS XACML. Core Specification: eXtensible Access Control Markup Language (XACML), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

Qi Li is a Ph.D. student in Department of Computer Science at Tsinghua University. His research interest includes network architecture and protocols, system and network security. Li has a M.S. in computer science from Chinese Academy of Sciences, China.

Xinwen Zhang is a research fellow in Samsung Information Systems America. His research interest includes computer and system security models, security in distributed and mobile computing systems, trusted computing and high assurance systems. Zhang has a Ph.D in information and software engineering from George Mason University.

Mingwei Xu is a professor in Department of Computer Science at Tsinghua University. His research interest includes computer network architecture, high-speed router architecture and network security. Xu has a Ph.D in computer science from Tsinghua University, China.

Jianping Wu is a professor of Department of Computer Science at Tsinghua University. He is also a director of Network Research Center of Tsinghua University. The major research areas of his group include next generation Internet architecture, terabit IP router, network management and security, P2P and overlay network, QoS management and QoS routing, formal methods and protocol testing. He is vice president of Internet Society of China (ISC), and chairman of APAN. Wu has a Ph.D in computer science from Tsinghua University, China.