Supporting Ad-hoc Collaboration with Group-based RBAC Model (Invited Paper)

Qi Li*, Xinwen Zhang†, Sihan Qing* Mingwei Xu‡
*Institute of Software, Chinese Academy of Sciences, Beijing 100080, China
Beijing ZhongkeAnsheng Corporation of Information Technology, Beijing 100080, China
Graduate School of Chinese Academy of Sciences, Beijing 100049, China
Email: liqi01@tsinghua.org.cn; qsihan@ercist.iscas.ac.cn
†Department of Information and Software Engineering, George Mason University, Fairfax, Virginia 22030, USA
Email: xzhang6@gmu.edu
‡Department of Computer Science, Tsinghua University, Beijing 100084, China
Email: xmw@csnet1.cs.tsinghua.edu.cn

***Abstract*— With the increasing accessibility of information and data, Role-Based Access Control (RBAC) has become a popular technique for security and privacy purposes. However, trusted collaboration between different groups in large corporate Intranets is still an unresolved problem. The challenge is how to extend existing access control model for efficient security management and administration to allow trusted collaboration between different groups. In this paper, we propose a group-based RBAC model (GB-RBAC) for this purpose. In particular, virtual group is proposed in our model to allow secure information and resource sharing in multi-group collaboration environments. All the members of a virtual group build trust relation between themselves and are authorized to join the collaborative work. The scheme and strategies provided in this paper meet the requirements of security, autonomy, and privacy for collaborations. As a result, our scheme provides an easy way to employ RBAC policies to secure ad-hoc collaboration.**

I. Introduction

Role-based Access Control(RBAC) [6] has emerged as a security technique for variant applications, and is the most attractive solution in intra-domain environments. However, with the development of these applications and the increasing information and data sharing between applications and domains, there are many security demands for collaborative work between different domains, and the original RBAC model cannot provide efficient authorization management in these environments.

The problem of collaboration in multi-domain environments is proposed by Gong *et al.* [1], and this work devotes to the solution of policy composition in distributed systems. Recently, several research efforts have been devoted to the topic of interoperation in multi-domain environments [7], [2], [4], [3]. In [3], Kapadia *et al.* propose a dynamic role translation model, and serval security issues are provided. Shafiq *et al.* [7], Piromrun *et al.* [4] and Joshi *et al.* [2] propose a series of secure interoperation schemes. In [2], Joshi *et al.* propose an XML based RBAC to specify multidomain policies. In [7], [4], solutions are proposed based on the Generalized Temporal Role Based Access Control Model (GTRBAC). In [7], 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, Piromrun *et al.* transform local GTRBAC policy to facilitate inter-domain interoperations [4]. 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.* [8] discuss access control requirements in collaborative systems and analyze existing access models including RBAC in collaborative environments.

In this paper, we propose a permission-driven collaboration scheme which utilizes the concept of virtual group in an advanced RBAC model called Group-based RBAC model (GB-RBAC). In this scheme, direct role mapping mechanism is eliminated for collaborations, thus many problems above mentioned disappear. With a novel administrative model of GB-RBAC, our scheme is an top-down approach to address the ad-hoc collaboration issue in distrusted environments and it also avoids the problem of SoD violations when integrating RBAC policies.

The paper is organized as follows. In Section II, a brief description of GB-RBAC and corresponding administrative model is presented. Section 3 explains the ad-hoc collaboration scheme with GB-RBAC, and Section 4 concludes this paper and presents some future work.

II. The GB-RBAC Model

The GB-RBAC model incorporates the component of groups into the RBAC96 [6] 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. Figure 2 shows the components of

[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 [3] .

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 [6]. Besides these, a GB-RBAC model 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).

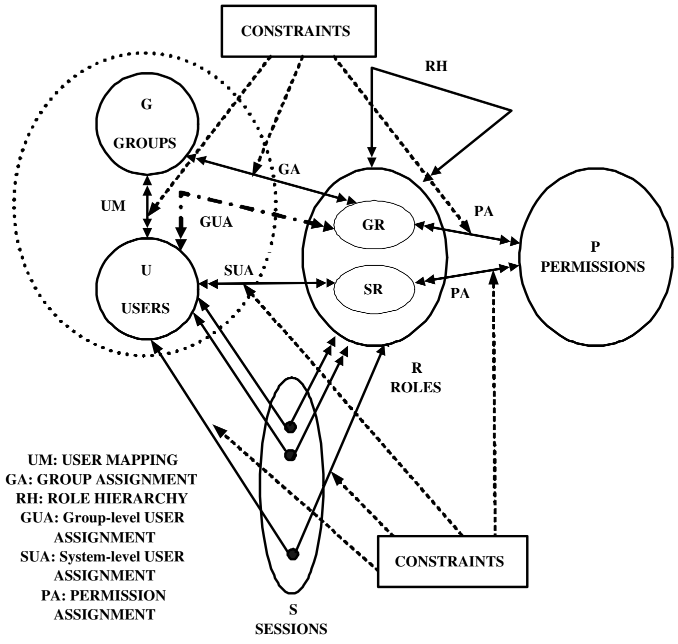


Fig. 1. GB-RBAC model

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).
- $R = SR \cup GR$, where $SR \cap GR = \emptyset$
- $PA \subseteq P \times R$, 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.
- $SUA \subseteq U \times SR$, system-level user-role assignment.
- $GUA \subseteq U \times GR$, group-level user-role assignment, and $(u, r) \in GUA$ only if $((u, g) \in UM) \wedge ((g, r) \in GA)$.
- $UA = SUA \cup GUA$, 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 $\bigcup_{r \in roles(s)} \{p | (\exists r'' \leq r)[(p, r'') \in PA]\}$

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 $\forall u, r, (u, g) \in UM \wedge r \in DSet(g) \rightarrow (u, r) \in GUA$. That is, a user who is mapped to a group obtains all the roles in the default role set of the group automatically.

In GB-RBAC, we propose two layers of roles through the group. In this way, a user assigned to some system-level roles (SR) can be assigned some group-level roles (GR) if he is affiliated with some groups. The user assigned to SR and GR gets different scopes of permissions. Besides the user-role assignment in the system scope, which is similar to the user-role assignment in URA97 [5], 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 the UM to a role in the GA, which is called group-level user-role assignment (GUA), while the previous 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 belongs to a group and the role is assigned to the group, and then the user holds the permissions to access resources defined with the group-level role. In addition, GB-RBAC provides $Dset$ through which a new user-role assignment mechanism is realized to reduce administrative tasks. In this way, a new member of a group can be assigned some default roles without administrator's involvement, and group administrators can assign other explicit roles to group members based on roles in $Dset$. A GB-RBAC model also can have constraints defined on many aspects shown in Figure 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 in GB-RBAC.

Two-level administration models referred as system-level and group-level administration model, respectively, are proposed to manage the relations defined in GB-RBAC. For these two administration levels, two types of administrative roles are defined in the administration model, called system-level administrative roles (SAR) and group-level administrative roles (GAR). These administrative roles also can form role hierarchies, respectively, similar to that of the regular roles in GB-RBAC. For simplicity we assume that $SAR \cap GAR = \emptyset$. The notion of prerequisite condition in different types of assignments is the key of our administrative model. There are three types of prerequisite conditions: user prerequisite conditions, permission prerequisite conditions, and group prerequisite conditions.

Definition 3: A *user prerequisite condition* is defined as a boolean expression using the usual $\wedge$ and $\vee$ operators on terms of the form x and $\overline{x}$, where x is a regular role (i.e., $x \in R$) or a group (i.g., $x \in G$). 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 $\overline{x}$ 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 membership of role(s) and group(s). As the membership of a role tests both SUA and GUA, the prerequisite condition define above is at least as expressive as that in URA97 [5]. Similarly, a permission prerequisite condition can be defined to test if a permission is assigned to a role or not. The set of all possible permission prerequisite conditions is denoted as CR_p.

Definition 4: A *group prerequisite condition* is defined as a boolean expression using the usual $\wedge$ and $\vee$ operators on terms of the form x and $\overline{x}$, 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 the GA relation to test the membership/nonmemberships of a group, which is used in the administration of group-role assignment.

Definition 5: System-level administrative grant model in GB-RBAC

- user-role assignment in SUA is controlled by means of the relation $can\_assign\_SUA \subseteq SAR \times CR_u \times 2^R$.
- permission-role assignment in SPA is controlled by the relation $can\_assignp\_PA \subseteq SAR \times CR_p \times 2^R$
- 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: Group-level administrative grant model in GB-RBAC

- 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/permission/group to be a member of role in role range $\{z\}$ if the user/permission/group satisfies the corresponding prerequisite condition y. Note that in a GB-RBAC model, users (e.g., user accounts), roles and permissions are created by the system administrators [2], and group administrators can manage their relations in the group level.

TABLE I
ADMINISTRATION CONTROL RULES

Type	Admin. Role	Rrereq. Condition	Group /Role Range
can_assign_UM	E-SSO	ER1	{@PRO1}
$can\_assignp\_PA$	E-SSO	PL1 $\wedge$ $\overline{QE1}$	[PE1, PE1]
can_assign_GUA	PM	@PRO1 $\wedge$ $\overline{QE1}$	{PE1}

As an example, consider a set of administration rules defined in the organization as Table I 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 administration domain. The roles are created as shown in the Figure 2: the figure above the dashed line presents the system level roles; the figure below the dashed line presents the group level roles. These two levels of roles both contain two types of roles: the normal roles such as resAA in the system level roles, PL1 in the group-level roles, and the administrative roles such as S-SSO in the system-level roles, GD in the group-level roles. Role hierarchy also exists among these roles in Figure 2. For example, there is a role hierarchy between the two system level roles: a junior-most role resAA and a senior-most role resAO. Between them, there are two other incomparable roles, resAD and resAM.

Now let us consider that Alice is a member of the system administrative role E-SSO, and Bob is a member of the role ED. According to rule can_assign_UM(E-SSO,ER1,{@PRO1}), Alice can assign Bob to group PRO1. Alice also can assign permission from the role PL1 assigned to PE1 since the rules $can\_assignp\_PA$(E-SSO,PL1 $\wedge\overline{QE1}$, [PE1, PE1]). Due to the space limitation, we skip the example of system-level role assignment rule which is similar with URA97 [5]. Now we change the scope to group level administrative rules. 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}) [3].

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

Definition 7: In system-level administration revocation model,

- user-role revocation in SUA is controlled by means of the relation $can\_revoke\_SUA \subseteq SAR \times 2^R$.
- permission-role revocation in PA is controlled by means of the relation $can\_revokep\_PA \subseteq SAR \times 2^R$.
- 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 $can\_revoke\_GA \subseteq SAR \times 2^R$.

[2]For simplicity, the administration model introduced in this paper does not include corresponding rules to create users, roles, and permissions, as well as role hierarchy administrations.

[3]The rules on GA is an application-specific issue, and this paper do not cover it.

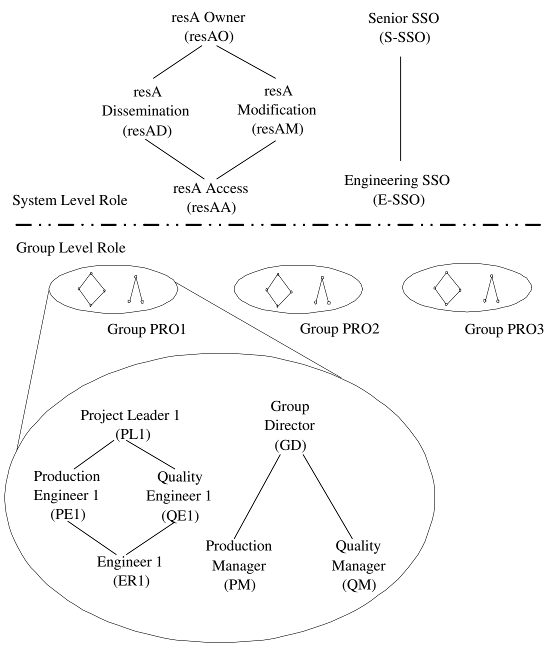


Fig. 2. Different level of roles in GB-RBAC

Definition 8: In group-level administration revocation model,

- The user-role revocation in GUA is controlled according to the relation $can\_revoke\_GUA \subseteq GAR \times 2^R$.

Specifically, there also are five types of $can\_revoke(x, z)$, which means an administrative member of role x can revoke a user, a member of permission or group for role (or group) z. Due to the space limitation, we skip the example of system-level role revocation rule which is similar with URA97 [5]. Let Alice be a member of E-SSO, and Bob be a member group PRO1 and role PE1. With rule can_revoke_UM(E-SSO,@PRO1), Alice is authorized to revoke membership of Bob from group PRO1. Through the rule $can\_revokep\_PA$(E-SSO, [ER1, PL1]), Alice can revoke permissions from any role range ER1 and PL1. Now we change the scope to group level administrative rules. The rule can_revoke_GUA(PM, (ER1,PL1)) indicates that Carol who is a member of PM can revoke Bob from PE1.

TABLE II
REVOCATION CONTROL RULES

Type	Admin. Role	Group/Role Range
can_revoke_UM	E-SSO	{@PRO1}
$can\_revokep\_PA$	E-SSO	[ER1, PL1]
can_revoke_GUA	PM	(ER1, PL1)

The GB-RBAC model and corresponding administrative model we discussed above are the fundamental work for our secure collaboration scheme. Our model not only simplifies the administrative tasks by the two level administrative model, but also provides a flexible administration for some dynamic application. For example, a serial of Video IP conferences are held in the scope of a group. Based on the permissions illustrated in Table III, 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. The member of PL1 not only has the permission to host this conference, but also can administrate GUA and change the these assignment through the group administrative rules above.

TABLE III
EXAMPLE OF ROLE ASSIGNMENT

Role	Permission	Role	Permission
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

III. Ad-hoc Collaboration Scheme in GB-RBAC

This section first identifies the generic access control requirements for ad-hoc collaborations, and then presents our solution with GB-RBAC.

A. 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 domain. A collaboration scheme should enable management autonomy in individual groups and information exchangeability between groups.

Previous work result in many violations/conflicts [7], [3] when a collaboration between different groups happens. Most of previous work implement collaborative through direct role map relations between different groups, where violations or problems happen such as user-specific SoD violation, role-specific SoD violation, role assignment violation [7] and role promotion [3]. In our work, we propose the concept of virtual group, and roles which are involved in collaborative work are exported into a virtual group from their original groups. In this way, most of violations/problems are eliminated such as the SoD violations mentioned above, and some constraints such as induced SoD are disposed at user-role assignment stage. However, our scheme have the following conflicts: conflicts of user names, role names and permission names; conflicts of permissions of roles in different groups. We analyze and solve these conflicts in the later of this section.

B. Collaboration Grant and Revocation in GB-RBAC

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 the links of the group-level components (roles and permissions) exported from the collaborative groups. Figure 3 illustrates application of VG.

In this paper, we focus on the ad-hoc collaboration administration in the group level administration model. Actually, the system-level administrators can also administrate collaborations. If the system-level administrators are involved in

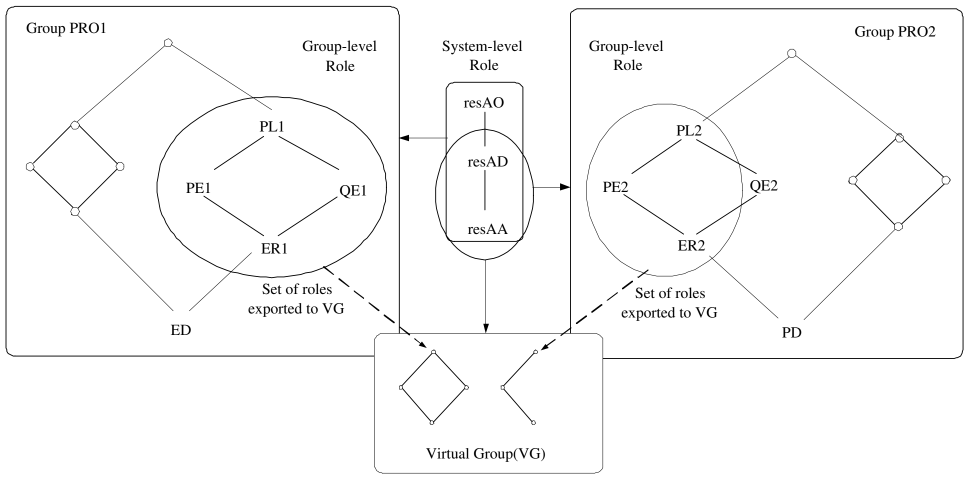


Fig. 3. Collaboration between groups with GB-RBAC

collaborative work, the procedure is simpler and it is more like an intra-domain work. So we discuss the scenario that the work only involves group-level administrators. In this paper, the scheme only involves group-level administrators. Collaboration building procedure is implemented as following steps: 1) A collaboration request is sent to the collaborative groups by an administrator of a group (called the VG founder), and the response is sent back to the founder; 2) The administrators of all the collaborative groups build VG using ColGrant Algorithm; 3) The 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 administration model in Section 2. 5) All the members of VG can start the collaborative work, and some collaboration modification can be realized using ColUpdate Algorithm. 6) The collaborative work finishes, and the administrators of normal groups who leave the VG last destroy the VG.

In a collaboration between groups, group-level permissions and roles can be exported into the virtual group. In Figure 3, PRO1 exports {ER1,PE1,QE1,PL1} to VG, and PRO2 exports {ER2,PE2,PL2} to VG. In this way, VG contains the links of roles {ER1,ER2,PE1,QE1,PE2,PL1,PL2} and the links of corresponding permissions. In this way, all the members assigned some specific roles in VG can act as appropriate roles in collaborative work respectively no matter where they come from.

For a group level collaboration scheme, there are three cases we should consider. Before we analyzing these three cases, we give the definition of role conflict which may exist when roles in different groups are exported to a virtual group, and the definition of role naming mechanism which is used in the process of building role collaboration.

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.

Conflict between r_i and r_j happens if there exist two permissions of r_i and r_j can be acquired by a user while two permissions can not be acquired by the user simultaneously, e.g., according to SoD constraints. For example, as illustrated in Figure 4(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 III). Although P1(conf1_speak) and P3(conf2_speak) can simultaneously achieved by a user (that is, the user can speak both at conference1 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 the Definition 9, there exists role conflict.

If a role conflict defined in Definition 9 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 the 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 role or permission 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.

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 conflicted role into two parts. The QE2 illustrated in Figure 4(c) conflicts with the role 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:

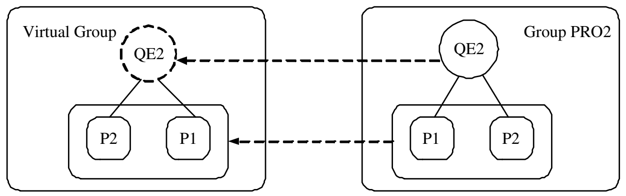
(1) Roles which can be directly exported into VG;

(2) Roles which can be partly exported into VG; That is, the subset of permissions acquired by the roles can be exported.

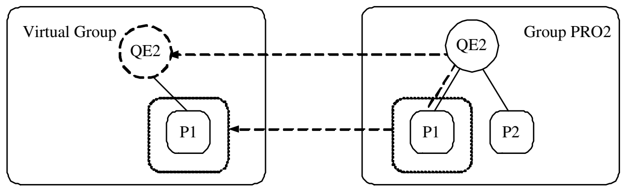
(3) Roles which should be wholly exported into VG. However, some role conflicts exist between these roles and existing roles in VG, and the set of permissions acquired by these roles should be exported individually.

Figure 4 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 also can directly export QE2. In the second case, only subset of permissions of QE2 can be exported to VG. Specifically, we need to split the permissions of QE2, and export the part with permission P1 to VG. In the third case, QE2 and PE1 conflicts in VG. Through the Definition 9, we should split the permissions of QE2 into the subsets P1 and P2. We then need to create two new roles QE21 and QE22 and add that roles to the set of P1 and P2. After that, QE21 and QE22 are be exported to VG, respectively, and P1 and P2 also can be exported to VG.

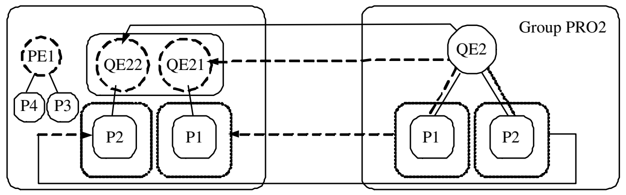
All users and permissions in a virtual group do not contain any genuine information. Actually, they are only link information which denotes which component comes from which group. However, a role in a virtual group is assigned with permissions of its linked role. The names of the components in virtual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



(a) All Export



(b) Partial Export



(c) Export in Presence of Role Conflict

Fig. 4. Role export scheme in GB-RBAC

Next, we present three algorithms which are used to build a virtual group, update the components of a virtual group, and destroy a virtual group, respectively. These algorithms consider three cases mentioned above when exporting roles and permissions of different groups to a virtual group. After a virtual group is established, assigning users to roles in the group can be performed by the group administrators following the user-role administration model presented in Section 2. So in this section we focus on the details of exportation of roles and permissions.

ColGrant algorithm shown in Figure 5 describes the main steps to build a virtual group among collaborative groups. In the process of collaboration building, one of the administrators [4] of 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 similar process, the 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 the $Dset$s of all the collaborative groups.

Now we consider some examples about ColGrant algorithm with Figure 4. 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 export procedures are implemented by the role-link function. We assume that $Dset$

[4]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$_y$ = ø
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. 5. ColGrant algorithm

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 2. In this way, the users in the virtual group can be authorized with permissions and start the collaborative work with each other.

We present two algorithms to update the components of a virtual group (ColUpdate) (see Figure 6) and delete a virtual group (ColRevo) (see Figure 7). 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 process of a role update, we unlink the role and re-export the updated role. In the ColRevo algorithm, we also need a flag to distinguish the three different cases. If a role is directly exported into a

```
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$_{set}$ ≠ ø
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. 6. ColUpdate algorithm.

virtual group, we delete the role link. However, if the role is split into 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.

We summary the main advantages of our scheme by comparing the previous work as follows: (1) Our collaboration scheme avoids infiltration and covert promotion problem [3]; (2) We propose a top-bottom approach to merge RBAC policies of different groups, thus less constraints are considered. Our top-bottom approach use a uniform naming mechanism

```
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. 7. ColRevo algorithm.

and two levels of roles to construct collaboration policies. In this way, the violations in previous work are eliminated in our scheme, and all constraints are leaved to be considered at user-role assignment stage. (3) We use virtual group to avoid direct role mapping, thus avoid many problems when integrating RBAC policies. In addition, the number limit of collaborative groups [4] is eliminated. (4) The permission-driven mechanism in our scheme simply reduces the conflicts of permissions.

IV. Conclusion and Future Work

In this paper, we propose an ad-hoc collaboration scheme in multi-group environment based on a group-based RBAC model. The scheme proposes a component of virtual group to enable secure collaboration 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 an easy and secure solution to support ad-hoc collaboration.

Several aspects need further study in our scheme. First of all, constraints for a virtual group need to be explored. Moreover, the proposed scheme needs to be extended to integrate advanced mechanisms and constraints to facilitate overall policy administration.

Acknowledgment

The work of S. Qing was supported by the Beijing Natural Science Foundation under Grant No. 4052016; the National Natural Science Foundation of China under Grant No. 60573042 and the National Grand Fundamental Research 973 Program of China under Grant No.G1999035802.

References

[1] L. Gong and X. Qian. Computational issues is secure interoperation. *IEEE Transactions on Software and Engineering*, 22(1):43–52, January 1996.

[2] J. Joshi, R. Bhatti, E. Bertino, and A. Ghafoor. Access control language for multidomain environments. *IEEE Internet Computing*, pages 40–50, Novermber-December 2004.

[3] A. Kapadia, J. AI-Muhtdai, R. Campbell, and D. Mickunas. IRBAC 2000: Secure interoperability using dynamic role translation. In *Technical Report: UIUCDCS-R-2000-2162*, 2000.

[4] S. Piromruen and J. Joshi. An RBAC framework for time constrained secure interoperation in multi-domain environments. In *Proceedings of 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems(WORDS'05)*, pages 36–48, 2005.

[5] R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of role. *ACM Transactions on Information and Systems Security*, 2(1):105–135, February 1999.

[6] R. Sandhu, E. Coyne, H. Reinstein, , and C.Youman. Role-based access control model. *IEEE Computer*, 29(2):38–47, February 1996.

[7] B. Shafiq, J. Joshi, E. Bertino, and A. Ghafoor. Secure interoperation in a multidomain environment employing RBAC poilcies. *IEEE Transactions on Knowledge and Date Engineering*, 17(11):1557–1577, Novermber 2005.

[8] W. Tolone, G. Ahn, T. Pai, and S. Hong. Access control in collaborative systems. *ACM Computing Surveys*, 37(1):29–41, May 2005.