Advanced Permission-Role Relationship in Role-Based Access Control*

Min Li[1], Hua Wang[1], Ashley Plank[1], and Jianming Yong[2]

[1] Department of Mathematics & Computing
University of Southern Queensland, Australia
Email: {limin, wang, plank}@usq.edu.au
[2] School of Information Systems, Faculty of Business
University of Southern Queensland, Australia
Email: yongj@usq.edu.au

Abstract. Permission-role assignment is an important issue in role-based access control (RBAC). There are two types of problems that may arise in permission-role assignment. One is related to authorization granting process. Conflicting permissions may be granted to a role, and as a result, users with the role may have or derive a high level of authority. The other is related to authorization revocation. When a permission is revoked from a role, the role may still have the permission from other roles. In this paper, we discuss granting and revocation models related to mobile and immobile memberships between permissions and roles, then provide proposed authorization granting algorithm to check conflicts and help allocate the permissions without compromising the security. To our best knowledge, the new revocation models, local and global revocation, have not been studied before. The local and global revocation algorithms based on relational algebra and operations provide a rich variety. We also apply the new algorithms to an anonymity scalable payment scheme.

1 Introduction

Role-based access control (RBAC) is a flexible and policy-neutral access control technology and is a promising access control technology for the modern computing environment [1, 3, 6, 16]. In RBAC, permissions(each permission is a pair of objects and operations) are associated with roles and users are assigned to appropriate roles thereby acquiring the roles' permissions. As such, a user in RBAC is a human being. It can be easily reassigned from one role to another. A role is a job function or job title and created for various job functions in an organization and users are assigned roles based on responsibilities and qualifications. A permission is an approval of a particular mode of access to one or more objects. The user-role and permission-role assignment relations are many-to-many between users and roles, and between roles and permissions as depicted in Fig.1. Roles can be granted new permissions as new applications come on

* The research is support by an ARC Discovery Grant DP0663414.

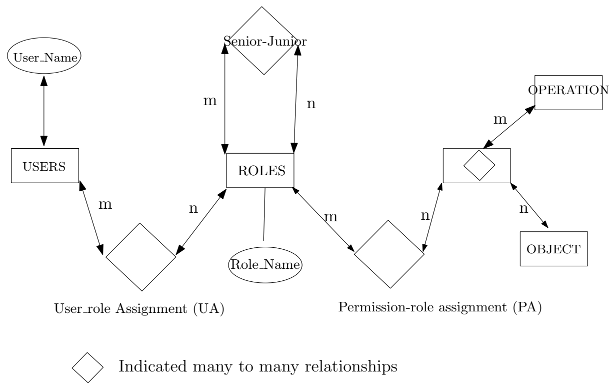

User_Name
Senior-Junior
OPERATION
USERS
ROLES
OBJECT
Role_Name
m
n
User_role Assignment (UA)
Permission-role assignment (PA)
Indicated many to many relationships

Fig. 1: RBAC relationship

line and permissions can be revoked from roles as needed. Within RBAC, users are not granted permission to perform operations on an individual object, but permissions are associated with roles.

Significant developments have been made within RBAC. The NIST model of RBAC [2] and Web implementation of RBAC incorporates an administrative tool that provides rudimentary support for an RBAC database that stores information about user and permission role assignments and role hierarchies [4]. Nyanchama and Osborn [7] define a role graph model that rigorously specifies operational semantics for manipulating role relations in the contexts of a role hierarchy. ARBAC97 builds on these previous attempts to construct administrative models [10] over all aspects of the RBAC model. Sandhu and Munawer [11] extends the ARBAC97 model by adding the concept of mobile and immobile permissions for the first time in this area. In [11], the authors distinguished two kinds of membership in a role. Immobile membership grants the role to have the permission, but does not make that permission eligible for further role assignment. Mobile membership on the other hand, covers both aspects.

However, there is a consistency problem when using RBAC management. For instance, if there are hundreds of permissions and thousands of roles in a system, it is very difficult to maintain consistency because it may change the authorization level, or imply high-level confidential information when more then one permission is requested and granted. Specifically, [11] does not mention conflicts when assigning permissions to roles. Therefore, there is no support to deal administrative role with regular roles, especially mobile and immobile members.

In this paper, we develop formal approaches to check the conflicts and therefore help allocate the permissions without compromising the security. We analyze authorization granting and revocation models with the mobility of permission-role relationship. Our main contribution in this paper is the relational algebra-based authorization granting and local, global revocation algorithms. Furthermore, we include an applicable example to illustrate our algorithms. Another

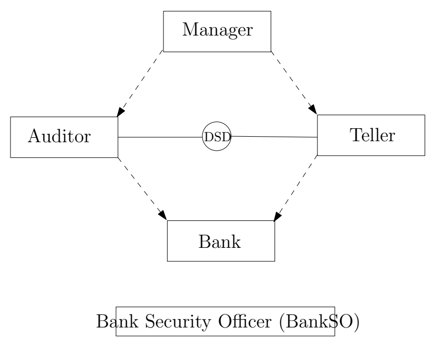


Fig. 2: Administrative role and role relationships in a bank

contribution is that our algorithms could check conflicts when granting more than one permission as mobile or immobile member to a role in the system. As far as we know, there is no previous work addressing these issues for permission allocation and conflict detection concerning on mobile memberships.

The organization of the paper is as follows. In section 2, we consider the mobility of permission-role relationship and problems related to permission assignment and revocation. The relational algebra-based authorization granting and revocation algorithms are given in section 3. In section 4, we review an anonymity scalable electronic commerce payment scheme and apply algorithms to this scheme. Comparisons with previous work are discussed in section 5. Finally, we conclude the paper in section 6.

2 Motivation and problem definitions

There are two kinds of membership between permissions and roles, namely mobile and immobile [11]. Immobile membership grants the role the permission but does not make that permission eligible for further role assignments. Mobile membership on the other hand covers both aspects which means the role has the permission and the permission also becomes eligible for assignment to other roles by appropriate administrative roles.

The distinction between mobile and immobile membership can be very important in practice. Fig. 2 shows the administrative and regular roles that exist in a bank department. The permission-role assignment allows us to give BankSo the authority to take a permission assigned to Manager and grant it to roles Teller, Auditor, and Bank. The idea is that each administrative role can delegate permissions of the senior role to more junior roles. While this may be acceptable for most permissions of a senior role, it is likely that some permissions are not suitable for such delegation. For instance in Fig. 2, suppose ‘approving a loan’ is a permission of the role Manger, which should only be executed by the Manger. Consider the two kinds of membership between permissions and roles, if this permission is assigned to the role Manger as a mobile member, it is possible that

PermName	Oper	Object	ConfPerm
Approval	approve	cash / check	Funding
Funding	invest	cash	Approval
Audit	audit	record	Teller
Teller	transfer	cash	Audit

Table 1: Example of the relation PERM

all the roles junior to the Manger can hold this permission through permission-role assignment, which leads to security breach. So this permission can only be assigned to Manager as immobile while the others can be assigned as mobile.

This example demonstrates that the situations with mobile and immobile relationship between permissions and roles can be very useful in practice to avoid the security breach. Throughout the paper, we consider the following two problems that may arise in permission-role assignment.

Authorization granting problem: Is a permission in conflict with the permissions of a role when granting the permission to the role as a mobile or immobile member?

Authorization revocation problem: Has a permission with mobile or immobile membership of a role been revoked from the role?

Conflicting permissions may be granted to a role in permission-role assignment. For example, the permission for approving a loan in a bank and that of funding a loan are conflicting. These two permissions can not be assigned to a role at the same time. It is easy to find conflicts between permissions when assigning permissions to a role in a small database but it is hard to find them when there are thousands of permissions in a system. Moreover, it is even more complicated if taking mobile and immobile permissions into account. Our aim is to provide relational algebra algorithms to solve these problems and then automatically check conflicts when assigning and revoking.

For convenience, we recall some basic definitions in paper [15] with no further explanation. Let D be a database with a set of relations REL and a set of attributes Attri. REL includes PERM, ROLE-PERM and SEN-JUN etc. Attri includes attributes such as Role-Name, PermName, Senior and Junior, etc.

PERM is a relation of PermName, Oper, Object and ConfPerm. Perm-Name is the primary key for the table and is the name of the permission in the system. Oper is the name of the operation granted. It contains information about the object to which the operation is granted. Object is the item that can be accessed by the operation, which may be a database, a table, a view, an index or a database package. ConfPerm is a set of permissions that is conflicting with the PermName in the relation. For example, a staff in a bank cannot have permissions of approval and funding at the same time (as well as permissions of audit and teller). The relation of PERM is expressed in Table 1.

SEN-JUN is a relation of roles in a system. SEN and JUN are the senior and junior of the two roles, senior roles are shown at the top of the role hierarchies. Senior roles inherit permissions from junior roles. For example, in Fig.2 role

Admin.role	Prereq.condition	Role Range
BankSO	Manager$\wedge\overline{Teller}$	[Auditor, Auditor]
BankSO	Manager$\wedge\overline{Auditor}$	[Teller, Teller]

Table 2: Example of *can-assignp-M* in Fig.2

Admin.role	Prereq.condition	Role Range
BankSO	Manager	[Auditor, Audior]
BankSO	Manager	[Teller, Teller]

Table 3: Example of *can-assignp-IM* in Fig.2

'Manager' is the senior role of role 'Teller' and inherit all permissions of 'Teller'. ROLE-PERM is a relation between the roles and the permissions, listing what permissions are granted to what roles. For example, permission 'Approval' is assigned to role 'Teller' and the permission 'Funding' to role 'Manager'.

3 Authorization granting and revocation algorithms based on relational algebra

In this section, we provide granting and revocation algorithms based on relational algebra. As discussed before, a permission's membership in a role can be mobile or immobile, so each role x is separated into two sub-roles Mx and IMx. Note that membership in Mx is mobile whereas membership in IMx is immobile.

A role x' has all permissions of a role x when $x' > x$ [3]. A permission p is an explicit member of a role x if $(p, x) \in PA$ and p is an implicit member of a role x if for some role $x' < x, (p, x') \in PA$. Combining mobile and immobile membership with the notion of explicit and implicit membership gives us four distinct kinds of role membership:

(1) Explicit mobile member $EMx = \{p|(p, Mx) \in PA\}$
(2) Explicit immobile member $EIMx$=$\{p|(p, IMx) \in PA\}$
(3) Implicit mobile member $ImMx = \{p|\exists x' < x, (p, Mx') \in PA\}$
(4) Implicit immobile member $ImIMx = \{p|\exists x' < x, (p, IMx') \in PA\}$

It is possible for a permission to have more than one kind of membership in a role at the same time. Hence there is strict precedence among these four kinds of membership [4].

$$EMx > EIMx > ImMx > ImIMx$$

A prerequisite condition is evaluated for a permission p by interpreting role x to be true if $p \in EMx \vee (p \in ImMx \wedge p \notin EIMx)$ and $\overline{x}$ to be true if $p \notin EMx \wedge p \notin EIMx \wedge p \notin ImMx \wedge p \notin ImIMx$.

[3] $x' > x$ means role x' is senior than x; $x' < x$ means role x' is junior than x.
[4] Even though a role can have multiple kinds of membership in a permission, at any time only one of those is actually in effect.

Admin.role	Prereq.condition	Role Range
BankSO	Bank	[Bank, Manager]

Table 4: *can-revokep-M* in Fig.2

Admin.role	Prereq.condition	Role Range
BankSO	Bank	[Bank, Bank]

Table 5: *can-revokep-IM* in Fig.2

For a given set of roles R, let AR be a set of administrative roles and CR denote all possible prerequisite conditions that can be formed using the roles in R. Not every administrator can assign a permission to a role. The following relations provide what permissions an administrator can assign mobile members or immobile members with prerequisite conditions.

Can-assignp-M, used to assign the permission as mobile members, is a relation in $AR \times CR \times 2^R$. While *can-assignp-IM* assigns the permission as immobile members. Table 2 and 3 show the example of these two relations. The meaning of $(BankSO, Manager \wedge \overline{Teller}, [Auditor, Auditor]) \subseteq$ *can-assignp-M* is that $BankSO$ can assign a permission whose current membership satisfies the prerequisite condition $Manager \wedge \overline{Teller}$ to role $Auditor$ as a mobile member. $(BankSO, Manager, [Teller, Teller]) \subseteq$ *can-assignp-IM* means that $BankSO$ can assign a permission whose current membership satisfies the prerequisite condition $Manager$ to role $Teller$ as an immobile member. To identify a role range within the role hierarchy, the following closed and open interval notation is used:

$$[x, y]=\{r \in R | x \geq r \wedge r \geq y\},\ (x, y]=\{r \in R | x > r \wedge r \geq y\}$$

$$[x, y)=\{r \in R | x \geq r \wedge r > y\},\ (x, y) = \{r \in R | x > r \wedge r > y\}$$

Suppose an administrator role (ADrole) wants to assign a permission p_j to a role r with a set of permissions P which may include mobile and immobile members. The permission p_j may be assigned as a mobile and immobile member if there is no conflict between p_j and the permissions in P. We analyze both mobile and immobile members in the following algorithm, which deals with whether the ADrole can assign the permission p_j to r with no conflicts. In algorithm 1, P^* is the extension of P, which includes the explicit and implicit members of P; i. e. $P^* = \{p | p \in P\} \cup \{p | \forall r' < r, (p, r') \in PA\}$.

Algorithm 1 provides a way to check whether or not a permission can be assigned as mobile or immobile member to a role. It can prevent conflicts when assign a permission to a role with mobile or immobile memberships as well. After considering the authorization, we consider the revocation of permission-role membership.

In the revocation model, a prerequisite condition is evaluated for a permission p by interpreting role x to be true if $p \in EMx \vee p \in EIMx \vee p \in ImMx \vee p \in ImIMx$ and $\overline{x}$ to be true if $p \notin EMx \wedge p \notin EIMx \wedge x \notin ImMx \wedge p \notin ImIMx$). Permission-role revocation of mobile and immobile memberships are authorized by the relations *can-revokep-M* $\subseteq AR \times CR \times 2^R$ and *can-revokep-IM* $\subseteq AR \times CR \times 2^R$ respectively.

$(BankSO, Manager, [Bank, Manager)) \subseteq$ *can-revokep-M* in Table 4 means that $BankSO$ can revoke the mobile membership of a permission from any role in

Algorithm 1: Authorization granting algorithm; *Grantp(ADrole, r, p_j)*.

Input: ADrole, role r and a permission p_j

Output: true if ADrole can assign p_j to r with no conflicts; false otherwise

```
Step 1. /*whether the ADrole can assign the permission $p_j$ to $r$ or not*/
    Let $S_{M1} = \pi_{prereq.condition}(\sigma_{admin.role=ADrole}(can\text{-}assignp\text{-}M))$,
        $S_{IM1} = \pi_{prereq.condition}(\sigma_{admin.role=ADrole}(can\text{-}assignp\text{-}IM))$,
    and $R = \pi_{Rolename}(\sigma_{Permname=p_j}(ROLE\text{-}PERM))$
    Suppose $p_j$ is a mobile member of the role $r$.
    If $S_1 = S_{M1} \cap R \neq \emptyset$, there exists a role $r_1 \in S_1$, such that $(p_j, r_1) \in PA$ and
      $r_1 \in \pi_{prereq.condition}(\sigma_{admin.role=ADrole}(can\text{-}assignp\text{-}M))$
    Go to Step 2;
    Suppose $p_j$ is a immobile member of the role $r$.
    If $S_2 = S_{IM1} \cap R \neq \emptyset$, there exists a role $r_2 \in S_2$, such that $(p_j, r_2) \in PA$ and
      $r_2 \in \pi_{prereq.condition}(\sigma_{admin.role=ADrole}(can\text{-}assignp\text{-}IM))$
    Go to Step 2;
    else
    return false and stop
Step 2. /* whether the permission $p_j$ is conflicting with permissions of $r$ or not*/
    Let $ConfPermS = \pi_{ConfPerm}(\sigma_{PermName=p_j}(PERM))$
    If $ConfPermS \cap P^* \neq \emptyset$, then $p_j$ is a conflicting permission with role $r$
    return false;
    else
    return true;
```

$[Bank, Manager)$ which satisfies the revoke prerequisite condition $Bank$. Similarly, the *can-revokep-IM* in Table 5 refers to revoking the immobile membership.

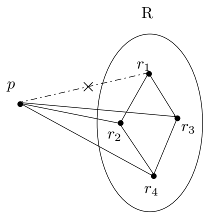


Fig. 3: Local revocation

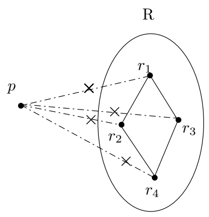


Fig. 4: Global revocation

Before giving out our revocation algorithms, first we introduce the concept of local and global revocation [12]. Local revocation only happens to the explicit relationship between permissions and roles, while global revocation effects all other roles which are junior to the role with the revoked permission. For local revocation, the permission is revoked only if the permission is an explicit member of the role. For example in Fig. 3, the role r_1 still has the permission p which has been locally revoked since the role is senior to role r_2 and r_3 which are associated

Algorithm 2: Local Revocation Algorithm; *Local-revoke(ADrole, r, p_j).*

```
Input: ADrole, role $r$ and a permission $p_j$
Output: true if ADrole can locally revoke $p_j$ from $r$; false otherwise
Step 1. If $p_j \notin \{p|(p,r) \in PA\}$
        return false and stop.
        /*there is no effect with the operation of the local revocation since the perm-
        ission $P_j$ is not an explicit member of the role r*/
        else Go to Step 2. /*$p_j$ is an explicit member of r*/
Step 2. /*whether the ADrole can revoke the permission $p_j$ from r or not*/
        Let $RoleRange1 = \pi_{RoleRange}(\sigma_{admin.role=ADrole}(can\text{-}revokep\text{-}M))$,
            $RoleRange2 = \pi_{RoleRange}(\sigma_{admin.role=ADrole}(can\text{-}revokep\text{-}IM))$
        and $Roles_{withp_j} = \pi_{RoleName}(\sigma_{PerName=p_j}(ROLE\text{-}PERM))$.
        Suppose $r \in EMp_j$
        If $r \in RoleRange1 \cap Roles_{withp_j} \neq \emptyset$; /*r is in the role range to be revoked
        by ADrole in can-revokep-M and the mobile membership with $P_j$*/
        return true;
        Suppose $r \in EIMp_j$
        If $r \in RoleRange2 \cap Roles_{withp_j} \neq \emptyset$; /*r is in the role range to be revoked
        by ADrole in can-revokep-IM and the immobile membership with $P_j$*/
        return true;
        else
        return false and stop. /*ADrole has no right to revoke the permission $P_j$
        from the role r*/
```

with the permission p. Therefore, local revocation from a role has no effect when a permission is an implicit member of the role. However, global revocation requires revocation of both explicit memberships and implicit memberships. If we globally revoke permission p from the role r_1, all the relationships between the permission p and roles junior to r_1 are revoked (see Fig. 4). Global revocation therefore has a cascading effect downwards in the role hierarchy. Global revocation of a permission's mobile and immobile membership from role r requires that the permission be removed not only from the explicit mobile and immobile membership in r, but also from explicit and implicit mobile and immobile membership in all roles junior to r.

Algorithms 2 and 3 are used to revoke permission $p_j \in P$ from a role r by ADrole, where P is the set of permissions which have been assigned to the role r. Algorithm 2 can be used revoke explicit mobile and immobile memberships, while Algorithm 3 can revoke explicit and implicit mobile and immobile members. It should be noted that the global revocation algorithm does not work if ADrole has no right to revoke p_j from any role in Jun.

4 Applying the relational algebra algorithms

In this section, we apply the new relational algebra algorithms to a consumer anonymity scalable payment scheme. We first briefly introduce the payment

Algorithm 3: Global Revocation Algorithm; *Global-revoke(ADrole, r, p_j).*

Input: ADrole, role r and a permission p_j

Output: true if ADrole can globally revoke p_j from r; false otherwise

```
Begin. If $p_j \notin P^*$
return false; /*there is no effect with the operation of the local revocation
since $p_j$ is not an explicit and implicit member of r*/
else
(1) If $p_j \in P$ is a mobile member of the role and $r \in EMp_j$,
Local-revoke(ADrole, r, $p_j$); /*$p_j$ is locally revoked as a mobile member*/
If $p_j \in P$ is an immobile member of the role and $r \in EIMp_j$,
Local-revoke(ADrole, r, $p_j$); /*$p_j$ is locally revoked as an immobile member*/
(2) Suppose $Jun = \pi_{junior}(\sigma_{Senior=r}(SEN\text{-}JUN))$
For all $y \in Jun$ with mobile membership with the permission
Local-revoke(ADrole, y, $p_j$) as $y \in EMp_j$;
For all $y \in Jun$ with immobile membership with the permission
Local-revoke(ADrole, y, $p_j$) as $y \in EIMp_j$;
/*$P_j$ is locally revoked from all such $y \in Jun$*/
If all local revocations are successful,
return true;
otherwise
return false.
```

scheme and consider the relationships of the roles in the scheme, and then analyze applications of our relational algebra algorithms.

4.1 The anonymity scalable electronic payment scheme

The payment scheme provides different degrees of anonymity for consumers. Consumers can decide the levels of anonymity. They can have a low level of anonymity if they want to spend coins directly after withdrawing them from the bank. Consumers can achieve a high level of anonymity through an anonymity provider (AP) agent without revealing their private information and are secure in relation to the bank because the new certificate of a coin comes from the AP agent who is not involved in the payment process.

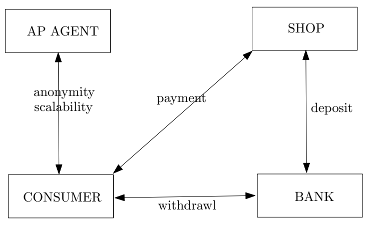
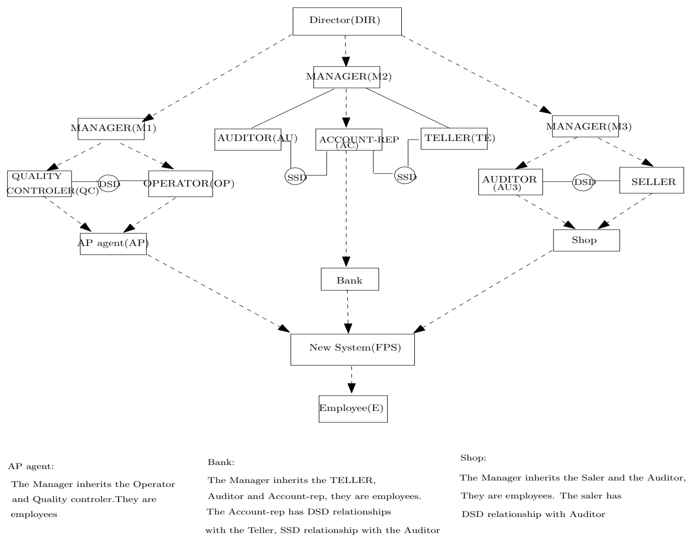


Fig. 5: Electronic cash model

RoleName	PermName
Director(DIR)	Funding
Director(DIR)	Approval
Director(DIR)	Teller
TELLER	Approval
FPS	Approval
Bank	Teller

Table 6: ROLE-PERM relation



AP agent:
The Manager inherits the Operator and Quality controler.They are employees

Bank:
The Manager inherits the TELLER, Auditor and Account-rep, they are employees. The Account-rep has DSD relationships with the Teller, SSD relationship with the Auditor

Shop:
The Manager inherits the Saler and the Auditor, They are employees. The saler has DSD relationship with Auditor

Fig. 6: User-role assignment in the payment scheme

Electronic cash has sparked wide interest among cryptographers [5, 8, 9]. In its simplest form, an e-cash system consists of three parts (a bank, a consumer and a shop) and three main procedures (withdrawal, payment and deposit). Besides the basic participants, a third party named anonymity provider (AP) agent is involved in the scheme. The AP agent helps the consumer to get the required anonymity but is not involved in the purchase process. The model is shown in Fig. 5. The AP agent gives a certificate to the consumer when he/she needs a high level of anonymity.

From the viewpoint of banks, consumers can improve anonymity if they are worried about disclosure of their identities. This is a practical payment scheme for internet purchases because it has provided a solution with different anonymity requirements for consumers. However, consumers cannot get the required level of anonymity if the role BANK and AP are assigned to one user. It shows the management importance of the payment scheme. To simplify the management, we analyze its management with the relational algebra algorithms.

4.2 Applying the authorization granting algorithm

Due to the length limit, we only include an application of the authorization granting algorithm. A hierarchy of roles and a hierarchy of administrative roles are show in Fig. 6 and 7 respectively, we define the *can-assignp-M* in Table 7. Here, we only show the process of assigning a permission to a role as a mobile member.

Here, we only analyze NSSO tuples in Table 7 (the analysis for APSO, BankSO and ShopSO are similar). The first tuple authorizes NSSO to assign permissions whose current membership satisfies the prerequisite condition role

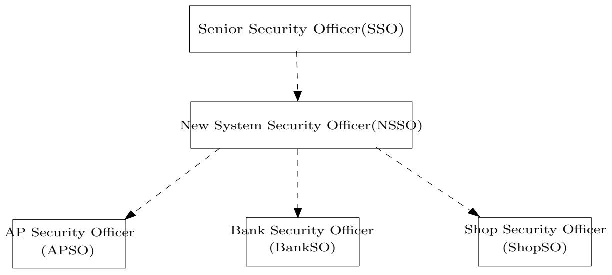

Senior Security Officer(SSO)
New System Security Officer(NSSO)
AP Security Officer (APSO)
Bank Security Officer (BankSO)
Shop Security Officer (ShopSO)

Fig. 7: Administrative role assignment in the scheme

Admin.role	Prereq.condition	Role Range
NSSO	DIR	[M1, M1]
NSSO	DIR	[M2, M2]
NSSO	DIR	[M3, M3]
APSO	M1 $\wedge \overline{OP}$	[QC, QC]
APSO	M1 $\wedge \overline{QC}$	[OP, OP]
BankSO	M1 $\wedge \overline{TE} \wedge \overline{AU}$	[AC, AC]
BankSO	M1 $\wedge \overline{TE} \wedge \overline{AC}$	[AU, AU]
BankSO	M1 $\wedge \overline{AU} \wedge \overline{AC}$	[TE, TE]
ShopSO	M1 $\wedge \overline{SALER}$	[AUDITOR, AUDITOR]
ShopSO	M1 $\wedge \overline{AUDITOR}$	[SALER, SALER]

Table 7: *Can-assignp-M* of Fig. 6

DIR to role M1 in the AP agent as mobile members. The second and third tuples authorize NSSO to assign permissions whose current membership satisfies the prerequisite condition role DIR to role M2 and M3 respectively as mobile members. Table 6 shows parts of the relations between permissions and roles in the scheme. Assume the role FPS with permission set $P = \{Approval\}$ and $P^* = P = \{Approval\}$. The administrative role NSSO wants to assign the permission $Teller$ to the role FPS as a mobile member. Using the first step of the granting algorithm *Grantp(NSSO, FPS, Teller)*, we could get:

$S = \pi_{prereq.condition}(\sigma_{admin.role=NSSO}(can\text{-}assignp\text{-}M)) = \{DIR\}$ and $R = \pi_{Rolename}(\sigma_{Permname=Teller}(ROLE\text{-}PERM)) = \{DIR, Bank\}$;
Since $R \bigcap S = \{DIR\} \neq \emptyset$, NSSO can assign permission $Teller$ to the role FPS as a mobile member. Applying the second step based on Table 1, we could get: $ConfPermS{=}\pi_{ConfPerm}(\sigma_{PermName=Teller}(PERM)) = \{Audit\}$ and $ConfPermS \cap P^* = \emptyset$. Hence there are no conflicts when assigning permission $Teller$ to the role FPS as a mobile member.

5 Comparisons

Our work substantially differs from [11] in two aspects. First, the paper [11] only introduce the definition of mobility of permission-role membership in permission-role assignment. By contrast, we discuss various cases in detail and focus on possible problems with mobility of permission-role relationship. Second, the authors only described the management of permission-role assignment with mobility in [11], but do not mention conflicts when assigning permissions to roles. Therefore, there is no support to deal administrative roles with regular roles in the proposal, especially mobile and immobile members. In this paper, we present a number of special authorization algorithms for access control, especially the local and global revocation algorithms which have not been studied before. These algorithms provide a rich variety of options that can handle the document of administrative roles with permissions as mobile and immobile members. In our earlier work [14], we developed authorization approaches for permission-role assignment. This paper is an extension of that study. Actually, if all membership is restricted to being mobile, our algorithms can imply the algorithms described in [14]. Moreover, compared with [14], mobile, immobile memberships and prerequisite conditions are discussed in this paper.

6 Conclusion

In this paper, we provide new authorization allocation algorithms for RBAC along with mobility that is based on relational algebra operations. The authorization granting algorithm, local and global revocation algorithm defined in this paper can automatically check conflicts when granting more than one permission as mobile or immobile member to a role in the system. We have also discussed how to use the algorithms for an electronic payment scheme.

References

1. Bertino, E., Ferrari, E. and Vijay Atluri. Specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1), February 1999.
2. Feinstein, H. L, et al. Small Business Innovation Research (SBIR): Role-Based Access Control: Phase 1, McLean, VA, SETA Corporation, January 20, 1995.
3. Ferraiolo, D. F., Barkley, J. F. and Richard Kuhn, D. A role based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security*, 2(1), February 1999.
4. Ferraiolo, D. F. and Barkley, J. F. Specifying and Managing Role-Based Access Control Within a Corporate Intranet. *Proc.of the 2ed ACM Workshop on Role-Based Access Control.* 1997, pp 77-82.
5. Frankel, Y., Tsiounis, Y. and Yung, M. Fair off-line e-cash made Easy. *in* Advance in Cryptology, *Proc. of Asiacrypt98*, LNCS 1294, PP. 257-270, 1998.
6. Gligor, V. D., Gavrila, S. T. and Ferraiolo, D. On the formal denition of separation-of-duty policies and their composition. *in Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 172-183, Oakland, CA, May 1998.
7. Nyanchama, M. and Osborn, S. The Role Graph Model and Conflict of Internet, A*CM Transaction on Information and System Security*, 2(1), 1999, pp. 3-33.
8. Okamoto. On efficient divisible electronic cash scheme, *in Advances in Cryptology-CRYPTO'95*, LNCS 963, Springer-Verlag, pp. 438-451, 1995.

9. Rivest, R. The MD5 Message-Digest Algorithm. RFC 1321. MIT Laboratory for Computer Science and RSA DATA Security. Inc, April 1992.
10. Sandhu, R., Bhamidipati, V. and Munawer, Q. The ARBAC97 model for role-based administration of roles, *ACM Transaction on Information and System Security*, 1(2), 1999, pp. 105-135.
11. Sandhu, R. and Munawer, Q. The ARBAC99 Model for Administration of Roles, *in the Annual Computer Security Applications Conference*, ACM Press, pp. 229-238, 1999.
12. Wang, H. and Cao, J. Delegating revocations and authorizations, accepted by the 1st International Workshop on Collaborative Business Processes, Brisbane, Australia, 2007
13. Wang, H., Cao, J. and Kambayashi, Y. Building a Consumer Anonymity Scalable Payment Protocol for the Internet Purchases, *The 12th International Workshop on Research Issues on Data Engineering: Engineering E-Commerce/E-Business Systems*, Pages: 159-168. Feb. 25-26, 2002, San Jose, USA.
14. Wang, H., Cao, J. and Zhang, Y. Formal authorization approaches for permission-role assignment using relational algebra operations, *Proceedings of the 14th Australasian Database Conference*, Feb. 2-7, 2003, Adelaide, Australia, Vol. 25, No.1, pages:125-134.
15. Wang, H., Cao, J. and Zhang, Y. Formal Authorization Allocation Approaches for Role-Based Access Control Based on Relational Algebra Operations, *The 3nd International Conference on Web Information Systems Engineering (WISE'2002)*, pages: 301-310. Dec. 3-6, 2002, Singapore.
16. Zurko, M., Simon, R. and Sanlippo, T. A user-centered modular authorization service built on an rbac foundation. *In Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 57-71, Oak-land, CA, May 1999.