Formal Authorization Allocation Approaches for Role-Based Access Control Based on Relational Algebra Operations

Hua Wang Jinli Cao Yanchun Zhang
Department of Maths & Computing
University of Southern Queensland
Toowoomba, QLD 4350, Australia
(wang, cao, zhang)@usq.edu.au

Abstract

In this paper, we develop formal authorization allocation algorithms for role-based access control (RBAC). The formal approaches are based on relational structure, and relational algebra and operations. The process of user-role assignments is an important issue in RBAC because it may modify the authorization level or imply high-level confidential information to be derived while users change positions and request different roles. There are two types of problems which may arise in user-role assignment. One is related to authorization granting process. When a role is granted to a user, this role may be conflict with other roles of the user or together with this role; the user may have or derive a high level of authority. Another is related to authorization revocation. When a role is revoked from a user, the user may still have the role from other roles.

To solve the problems, this paper presents an authorization granting algorithm, and weak revocation and strong revocation algorithms that are based on relational algebra. The algorithms can be used to check conflicts and therefore to help allocate the roles without compromising the security in RBAC. We describe how to use the new algorithms with an anonymity scalable payment scheme. Finally, comparisons with other related work are discussed.

1. Introduction

Recently, role based access control (RBAC) has been widely used in database system management and operating system products. RBAC involves individual users being associated with roles as well as roles being associated with permissions (Each permission is a pair of objects and operations). As such, a role is used to associate users and permissions. A user in this model is a human being. A role is a job function or job title within the organization associated with authority and responsibility.

Permission is an approval of a particular operation to be performed on one or more objects. As shown in Figure 1, the relationships between users and roles, and between roles and permissions are many-to-many. (i.e. a permission can be associated with one or more roles, and a role can be associated with one or more permissions). The security policy of the organization determines role membership and the allocation of each roles capabilities.

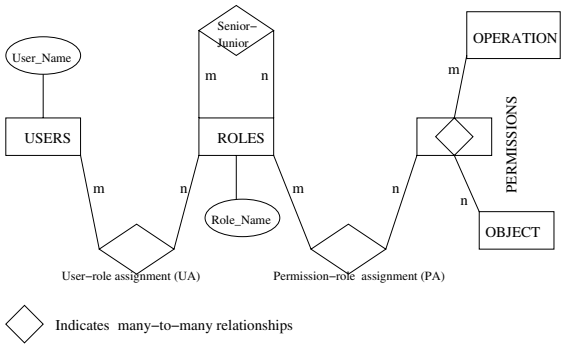

Figure 1. RBAC relationship

In 1993, the National Institute of Standards and Technology (NIST) developed prototype implementations, sponsored external research [6], and published formal RBAC models [7, 9]. Many organizations prefer to centrally control and maintain access rights, not so much at the system administrator's personal discretion but more in accordance with the organization's protection guidelines [5]. RBAC is being considered as part of the emerging SQL3 standard for database management systems, based on their implementation in Oracle 7 [13]. Many RBAC practical applications have been implemented [3, 8, 14].

However, there is a consistency problem when using RBAC management. For instance, if there are hundreds of roles and thousands of users in a system, it is very dif-

Proceedings of the 3rd International Conference on Web Information Systems Engineering (WISE'02)
0-7695-1766-8/02 $17.00 © 2002 IEEE

ficult to maintain consistency because it may change the authorization level, or imply high-level confidential information to be derived when more than one role is requested and granted.

We develop formal approaches to check the conflicts and therefore help allocate the roles without compromising the security. The formal approaches are based on relational structure, relational algebra, which has a set of complete and sound axioms. As far as we know, this is the first kind of work in this area to address the formal approaches for role allocation and conflict detection.

The paper is organized as follows. In the next section, we identify the problems related to role assignment and revocation. Relational algebra-based authorization granting algorithm and revocation algorithms are developed in section 3. In section 4, we review an anonymity scalable electronic commerce payment scheme. We then apply the formal authorization approaches to this scheme in section 5. Comparisons with related work are discussed in section 6 and the conclusions are in section 7.

2. Problem Definitions

With RBAC, users cannot associate with permissions directly but roles. Permissions must be authorized for roles, and roles must be authorized for users. The RBAC security model has two components: MC_0 and MC_1 [14]. Model component MC_0, called the RBAC authorization database model, defines the RBAC security properties for authorization of static roles. Static properties of an RBAC authorization database include role hierarchy, inheritance, cardinality, and static separation of duty. MC_1 called the RBAC activation model, defines the RBAC security properties for dynamic activation of roles. Dynamic properties include role activation, permission execution, dynamic separation of duties, and object access. In particular, the RBAC model supports the specification of several aspects.

a. User/role associations – the constraints specifying user authorizations to perform roles;

b. Role hierarchies – the constraints specifying which role may inherit all of the permissions of another role;

c. Duty separation constraints – these are role/role associations indicating conflict of interest:

c1. Static separated duty (SSD) – a constraint specifying that a user cannot be authorized for two different roles;

c2. Dynamic separated duty (DSD) – a constraint specifying that a user can be authorized for two different roles but cannot act simultaneously in both;

d. Cardinality – the maximum number of users allowed, i.e. how many users can be authorized for any particular role (role cardinality), e.g., only one manager.

A comprehensive administrative model for RBAC must account for all four issues mentioned above, among others. However, user-role assignment is a particularly critical administrative activity. This is because assigning people to tasks is a normal managerial function and assigning users to roles is a natural part of assigning users to tasks. Therefore this paper will focus on user-role assignment.

Let D be a database with a set of relations REL, a set of attributes A. REL includes ROLES, USERS, Can-assign, Can-revoke, SEN-JUN, and Role-User etc. A includes attributes such as RoleName, UserID, UserName etc from the relations. R_1 is a set of roles $R_1 = \{r_1, r_2, ..., r_n\}$; U is a set of users $U = \{u_1, u_2, ..., u_l\}$. Roles are in two categories, one is administrative roles (admin.role), the other is regular roles (role) that need to be assigned to or revoked from users by administrative roles. The roles assigned to a user by administrators may be in conflicts. For example, SSD and DSD relationships may not be sympathetic with the roles associated with the user; on the other hand, because of role hierarchies, the user may still have the permissions of a role which has been revoked. In the latter case, the user is able to access objects and has operations on the objects. The problems arising in processes of assigning and revocation are evident.

Authorization granting problem – How to check whether a role is in conflict with the roles of a user?

Authorization revocation problem – How to find whether permissions of a role have been revoked from a user or not?

For example, Figure 2 shows a system administrative role (ShopSO) in a shop to manage regular roles such as SELLER, AUDITOR and MANAGER. Role MANAGER inherits SELLER and AUDITOR. SELLER has a DSD relationship with AUDITOR. The administrative role ShopSO can assign a user to be AUDITOR or SELLER but not both simultaneously, otherwise it compromises the security of the shop system. It is easy to find conflicts between roles when assigning roles to a user in a small database system but it is hard to find them when there are thousands of roles in a system. Our aim is to provide relational algebra algorithms to solve the problems and then automatically check conflicts when assigning and revoking.

Some relations in set REL are detailed below.

ROLES - This relation has $(n + 1)$ attributes when there are n roles.

The first attribute, RoleName is the primary key for the relation, and represents the name of a role. From the second attribute to $(n + 1)th$ attribute, it describes the state of conflicts with the RoleName in the relation and its domain is {-1, 0}, where '-1' means conflicting with the RoleName and '0' means not.

The ROLES relation in Figure 2 is in Table 1. The attribute SELLERC shows whether the role SELLER is conflicting with the RoleName in the relation or not. For instance, in the third tuple, a user with role SELLER has conflicts with the role AUDITOR.

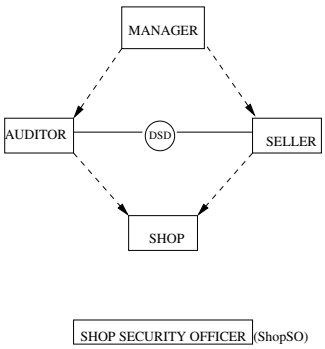

Figure 2. Administrative role and role Relationships in a shop

RoleName	MANAC	AUDC	SELLERC	SHOPC
MANAGER	0	0	0	0
AUDITOR	-1	0	-1	0
SELLER	-1	-1	0	0
SHOP	-1	-1	-1	0

Table 1. The relation ROLES in Figure 1

USERS - This relation has two attributes {UserID, UserName }, UserID is the identity of a user and UserName, which domain is the set of a list of users in the system. UserID and UserName are recorded for each user. UserName is the primary key for the table. For example, there are two users David and Tony in the system of Figure 2. The USERS relation is in Table 2.

UserID	UserName
0001	David
0002	Tony

Table 2. The relation USERS in Figure 1

Roles are managed by administrative roles. Senior roles are shown at the top of the hierarchies. Senior roles inherit permissions from junior roles. Let $x > y$ denote x is senior to y with obvious extension to $x \geq y$.

SEN_JUN - This is a relation of roles in a system. Senior is the senior of the two roles. Table 3 expresses the SEN_JUN relationship in Figure 2.

ROLE_USER - defines a relationship between USERS and ROLES {RoleName, UserName}.

RoleName is a foreign key RoleName in the ROLES. It explains a role is assigned to a user.

UserName is a foreign key UserName from the USERS.

Suppose MANAGER is assigned to user Tony and SELLER to user David, Table 4 expresses the ROLE_USER relationship.

Senior	Junior
MANAGER	AUDITOR
MANAGER	SELLER
MANAGER	SHOP
SELLER	SHOP
AUDITOR	SHOP

Table 3. SEN_JUN table in Figure 1

RoleName	UserName
MANAGER	Tony
SELLER	David

Table 4. ROLE_USER table

It is easy to know a role set associated with a user from ROLE_USER. Therefore, the authorization granting problem can be changed to whether a role is conflicting with a set of roles R or not.

3. Authorization granting and revocation algorithms based on relational algebra

We will develop granting and revocation algorithms based on relational algebra in this section. The notion of a prerequisite condition, Can-assign and Can-revoke is a key part in the processes of user-role assignment [15].

A **prerequisite condition** is an expression using boolean operators $\wedge$ and $\vee$ on terms of the form x and $\bar{x}$ where x is a role and $\wedge$ means "and ", $\vee$ means "or ". A prerequisite condition is evaluated for a user u by interpreting x to be true if $(\exists x' \geq x), (u, x') \in UA$ and $\bar{x}$ to be true if $(\forall x' \geq x), (u, x') \notin UA$, where UA is a set of user-role assignments. $\diamond$

For a given set of roles R let CR denote all possible prerequisite conditions that can be formed using the roles in R. Not every administrator can assign a role to a user. The following relation provides what roles an administrator can assign with prerequisite conditions.

Can-assign is a relation of $\subseteq AR \times CR \times 2^R$, where AR is a set of administrative roles. $\diamond$

User-role assignment (UA) is authorized by Can-assign relation. Table 5 shows the can-assign relations with the prerequisite conditions in the example.

The meaning of Can-assign $(ShopSO, SHOP, \{SELLER, AUDITOR\})$ is that a member of the administrative role ShopSO can assign a user whose current membership satisfies the prerequisite condition SHOP to be a member of roles SELLER and AUDITOR. To identify a role range within the role hierarchy, the following closed and open interval notation is used.

Admin.role	Prereq.Condition	Role Range
ShopSO	SHOP	[SHOP, SHOP]
ShopSO	SHOP $\wedge \overline{SELLER}$	[AUDITOR, AUDITOR]
ShopSO	SHOP $\wedge \overline{AUDITOR}$	[SELLER, SELLER]
ShopSO	SELLER $\wedge$ AUDITOR	[MANAGER, MANAGER]

Table 5. Can-assign in the example

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

Supposing an administrator role ADrole wants to assign a role r_j to a user with a set of roles R. R^* is an extension of R, $R^* = \{x | x \in R\} \cup \{x | \exists x' \in R, x' > x\}$.

Authorization granting algorithm Grant(ADrole, R, r_j)
Input: ADrole, a set of roles R and a role r_j.
Output: true if ADrole can assign role r_j to R with no conflicts; false otherwise.
Begin:
Num: = 0
Step 1. /* *Whether the ADrole can assign the role r_j to R or not* */

Suppose $S_1 = R^* \cap S_2$ where

$S_2 = \pi_{Prereq.Condition}(\sigma_{admin.role=ADrole}(Can - assign))$

if $S_1 \neq \phi$,
then there exists role $r \in S_1$, such that
$r_j \in \pi_{RoleRange}(\sigma_{\{ADrole,r\}}(Can - assign))$
go to step 2 /* *r_j is in the range to be assigned by ADrole in Can-assign* */
else
return false and stop. /**the admini.role has no right to assign the role r_j to R^** */
Step 2. /**whether the role r_j is conflicting with roles in R or not*/

for each role r_i in R, do
$Num_i = \pi_{r_jC}(\sigma_{RoleName=r_i}(ROLES))$

/* *r_jC is an attribute that describes conflicting states of r_j with other roles* */
if $Num_i = -1$
r_j is a conflicting role with R, *return* false;
if for all $r_i \in R$, $Num_i = 0$ *return* true.
/* *r_j is not a conflicting role with R* */ ◇

Theorem 1 The authorization granting algorithm can prevent conflicts when assigning a role to a user.

Admin.role	Role Range
ShopSO	[Shop, MANAGER)

Table 6. Can-revoke

Proof Assuming an administrator role ADrole wants to assign a role r_j to a user which associates with a role set R. While step 1 in the algorithm has checked whether the ADrole can assign the role r_j to a user or not, the second step has decided whether the role r_j is conflicting with roles in R or not. Indeed, r_j can be assigned to the user if for all $r_i \in R$, $Num_i = 0$. Otherwise r_j is a conflicting role with R. ◇

The authorization granting problem is solved by the authorization granting algorithm. Computing S_1 in the first step takes time proportional to n if n is presented as the number of roles. Computing $\sigma_{admin.role=ADrole}(Can - assign)$ and $\pi_{Prereq.Condition}(\sigma_{admin.role=ADrole}(Can - assign))$ needs time $O(n)$. Thus, the step 1 takes time $O(n^2)$. In the second step, the computations of $\sigma_{RoleName=r_i}(ROLES)$ and $\pi_{r_jC}(\sigma_{RoleName=r_i}(ROLES))$ spent $O(n)$ and $O(1)$ respectively. It needs time $O(n^2)$ to get Num_i for all $r_i \in R$. Therefore the total time spent in the authorization granting algorithm is proportional to n^2.

Corollary 1 The authorization granting algorithm has time complexity $O(n^2)$ for the case of n roles in a system.

Now we consider revocation of user-role membership. Similar to Can-assign relation in granting a role to a user, there is a Can-revoke relation between administrative roles and regular roles.

A relation $\mathbf{Can - rev\ ok} \subseteq AR \times 2^R$ shows which role range administrative roles can revoke, where AR is a set of administrative roles. ◇

The meaning of Can-revoke $(ShopSO, [Shop, MANAGER))$ in Table 6 is that a member of the administrative role Shop can revoke membership of a user from any role in [Shop, MANAGER).

Due to role hierarchy, a role x' has all permissions of role x when $x' > x$. A user with two roles $\{x', x\}$ still has the permissions of x if only to revoke x from the user. The explicit member of a role x is a set of user $\{U | (U, x) \in UA\}$ and the implicit member of role x is a set of user $\{U | \exists x' > x, (U, x') \in UA\}$. To solve the authorization revocation problem, we need to revoke the explicit member of a role first if a user is an explicit member, then revoke the implicit member.

Following are two algorithms for revocation of a role r_j from a set of roles R by an administrative role ADrole, where R is a set of roles which are assigned to a user. The

first one is weak revocation algorithm and another one is strong revocation algorithm. The weak revocation only revokes explicit membership from a user and does not revoke implicit membership but the strong revocation revokes both explicit and implicit members.

Weak revocation Algorithm Weak_revoke(ADrole, R, r_j)
Input: ADrole, a set of roles R and a role r_j.
Output: true if ADrole can weakly revoke role r_j from R; false otherwise.
Begin:
if $r_j \notin R$,
return false; /* *there is no effect with the operation of the weak revocation since the user is not an explicit member of* r_j */
else /* *The user with the role set* R *is an explicit member of* r_j */
1. *if* $r_j \in \pi_{RoleRange}(\sigma_{admin.role=ADrole}(Can-revoke))$,
return true; /**the user's explicit membership in* r_j *is revoked* */
2. *if* $r_j \notin \pi_{RoleRange}(\sigma_{admin.role=ADrole}(Can-revoke))$,
return, false. /* *ADrole has no right to revoke the role* r_j *from the user* */ ◇

We have the following result with the weak revocation algorithm.

Theorem 2 A role r_j is revoked by the weak revocation algorithm $Weak\_revoke(ADrole, R, r_j)$ if the user is an explicit member of role r_j and the ADrole has the right to revoke r_j from the Can-revoke relation. ◇

It takes time $O(n)$ to check if $r_j \notin R$ when there are n roles in a system. The computations of $\sigma_{admin.role=ADrole}(Can - revoke)$ and $\pi_{RoleRange}(\sigma_{admin.role=ADrole}(Can - revoke))$ take time $O(n)$. To check whether

$$r_j \in \pi_{RoleRange}(\sigma_{admin.role=ADrole}(Can-revoke))$$

needs time $O(n)$. Thus, the time complexity of the Weak revocation algorithm is $O(n)$.

Corollary 2 Weak revocation algorithm has time complexity $O(n)$ when there are n roles in a system.

A user still has permissions of a role which has been weakly revoked if a role associated with the user seniors the role revoked. To solve the authorization revocation problem, we need strong revocation which requires revocation of both explicit and implicit membership. Strong revocation of a user's membership in x requires that the user be removed not only from explicit membership in x, but also from explicit and implicit membership in all roles senior to x. Strong revocation therefore has a cascading effect upwards in the role hierarchy.

Strong revocation algorithm Strong_revoke(ADrole, R, r_j)
Input: ADrole, a set of roles R and a role r_j.
Output: true, if it can strong revoke role r_j from R; false otherwise.
Begin:
if $r_j \notin R^*$,
return false; /* *there is no effect of the strong revocation since the user is not an explicit and implicit member of* r_j */
else,
1. if $r_j \in R$, do Weak_revoke(ADrole, R, r_j);
/**r_j is weakly revoked from R**/;
2. Suppose
$Sen = R^* \cap \pi_{Senior}(\sigma_{Junior=r_j}(SEN - JUN))$,
for all $y \in Sen$, do Weak_revoke(ADrole, R, y);
/**the user is weakly revoked from all such* $y \in Sen$*/.
If all the weak revocations are successful,
return true;
otherwise, *return* false. ◇

It should be noted that Weak_revoke(ADrole, R, r_j) and Weak_revoke(ADrole, R, y) do not work if ADrole has no right to revoke r_j. We have the following consequence.

Theorem 3 The explicit and implicit member of role r_j are revoked from the user by the Strong revocation algorithm $Strong\_revoke(ADrole, R, r_j)$ if the ADrole has the right to revoke r_j from the Can-revoke relation.

Corollary 3 The authorization revocation problem is solved by the Weak revocation algorithm and Strong revocation algorithm.

It needs $O(n)$ to check whether $r_j \notin R^*$ if there are n roles in a system. The computations of $\sigma_{Junior=r_j}(SEN - JUN)$ and $\pi_{Senior}(\sigma_{Junior=r_j}(SEN - JUN))$ takes time proportional to m where m is the number of tuples in the relation SEN-JUN and $m < n$. It takes time proportional to $n * m$ to compute Sen. Other operations $r_j \in R$, r_j weak revocation and $y \in Sen$ takes time $O(n)$. Therefore the total time spent with the Strong revocation algorithm is $O(n^2)$.

Corollary 4 The Strong revocation algorithm has time complexity $O(n^2)$ when there are n roles in a system.

In the remaining parts of this paper, the new relational algebra approaches will be used with a payment scheme. We review the payment scheme first.

4. Review of the anonymity scalable electronic payment scheme

Algorithms will be used in a consumer anonymity scalable 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. The scheme is briefly reviewed below [18].

Electronic cash has sparked wide interest among cryptographers ([12, 20, 10, 17], etc.). 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 will help the consumer to get the required anonymity but will not be involved in the purchase process. The model is shown in Figure 3. The AP agent gives a certificate to the consumer when s/he needs a high level of anonymity.

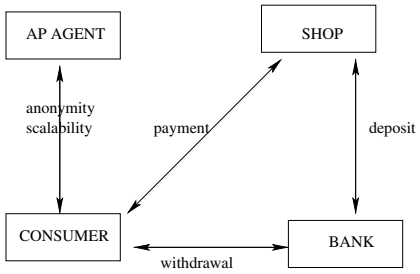

Figure 3. Electronic cash model

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 a user. It means the management of the payment scheme is an important issue. To simplify the management of the scheme and prevent errors in the user-role assignment, we will analyze its management with the relational algebra algorithms.

5. Applications of the relational algebra algorithms

The new relational algebra algorithms will be applied to the payment scheme in this section. We first consider the relationships of the roles in the scheme, then analyze applications of the relational algebra algorithms.

5.1 Duty separation constraints

There are two types in duty separation constraints. One is SSD and another one is DSD. In Figure 4, for example,

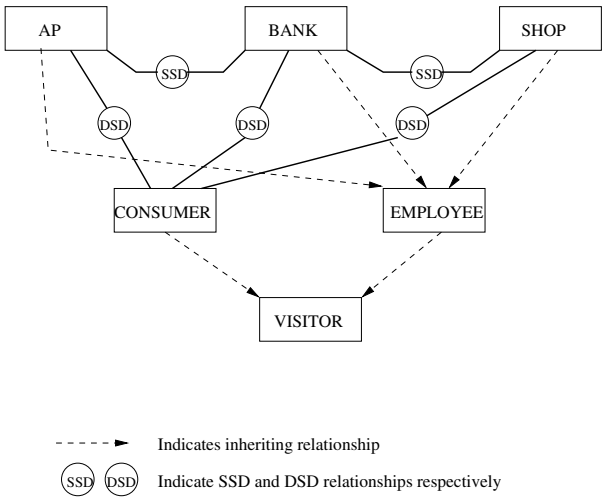

Figure 4. The relationships of the roles in the scheme

since all staff in the AP agent, the bank and the shop are employees, their corresponding roles inherit the employee role. The role AP, SHOP and BANK have DSD relationships with the role CONSUMER. This indicates that an individual consumer cannot act the roles of AP, SHOP or BANK simultaneously. The staff in these three participants have to first log out if they want to register as consumers. For example, a consumer, who is a staff member of the AP agent and is able to act the role AP, can ask the AP agent to help him to get a coin with a high level anonymity. But as a consumer, s/he cannot give herself/himself a new certificate of a coin when s/he works for the AP agent. Another staff member of the AP agent should do the job for this person.

The role AP has an SSD relationship with BANK. This is because the duty of the AP is to help a consumer to get a coin with a high level of anonymity. The BANK knows the old coin and its certificate. The AP sends the new certificate of the new coin to the consumer. The role BANK will know the new certificate and new coin if the same staff member from the AP agent and the bank processed the coin for the consumer. If this occurs, the consumer cannot have a coin with the required anonymity because the BANK has known the new coin. The SHOP also has an SSD relationship with the BANK since the BANK verifies the payment as well as depositing the coin to the shop's account. The SSD relationship is also a conflict of interest relationship like the DSD relationship but much stronger. If two roles have a SSD relationship, then they may not even be authorized to the same individual. Thus, the role AP, BANK, and SHOP may never be authorized to the same individual.

5.2 An application of the authorization granting algorithm

A hierarchy of roles and a hierarchy of administrative roles are shown in Figure 6 and Figure 5 respectively. Table 7 shows the Can-assign relations with the prerequisite conditions in the scheme and Table 10 shows ROLES relationship (the attribute M1C, for example, shows whether the role M1 is conflicting with the RoleName in the relation or not).

Let us consider the APSO tuples in Table 7 (the analysis for BankSO and ShopSO are similar). The first tuple authorizes APSO to assign users with the prerequisite role FPS into members in the AP agent (AP). The second one authorizes APSO to assign users with the prerequisite condition $FPS \wedge \overline{OP}$ to be quality controllers (QC). Similarly, the third tuple authorizes APSO to assign users with the prerequisite condition $FPS \wedge \overline{QC}$ to be operators (OP). The second and third tuple show that the APSO can grant a user who is a member of the AP agent into one but not both of QC and OP. This illustrates how mutually exclusive roles can be forced. However, for the NSSO and SSO these are not mutually exclusive. The fourth tuple authorizes APSO to put a user who is a member of both QC and OP into a manager (M1). Of course a user could have become a member of both QC and OP only by actions of a more powerful administrator than APSO.

Admin.role	Prereq.Condition	Role Range
APSO	FPS	[AP, AP]
APSO	FPS $\wedge\overline{OP}$	[QC, QC]
APSO	FPS $\wedge\overline{QC}$	[OP, OP]
APSO	QC $\wedge$ OP	[M1, M1]
BankSO	FPS	[Bank, Bank]
BankSO	FPS $\wedge\overline{TE} \wedge \overline{AU}$	[AC, AC]
BankSO	FPS $\wedge\overline{TE} \wedge \overline{AC}$	[AU, AU]
BankSO	FPS $\wedge\overline{AU} \wedge \overline{AC}$	[TE, TE]
BankSO	TE $\wedge AU \wedge$ AC	[M2, M2]
ShopSO	FPS	[Shop, Shop]
ShopSO	FPS $\wedge\overline{SELLER}$	[AUDITOR, AUDITOR]
ShopSO	FPS $\wedge\overline{AUDITOR}$	[SELLER, SELLER]
ShopSO	SELLER $\wedge$ AUDITOR	[M3, M3]
NSSO	FPS	(FPS, DIR)
SSO	E	[FPS, FPS]
SSO	FPS	(FPS, DIR]

Table 7. Can-assign relation in the scheme

Assume Bob is a user with role FPS. The administrative role NSSO wants to assign the role AP to Bob. Using the granting algorithm *Grant(NSSO, FPS, AP)*, the first step, $R^* = R = \{FPS\}$ and

$$\pi_{Prereq.Condition}(\sigma_{Admin.role=NSSO}(Can-assign)) = \{FPS\},$$

then

$$R^* \cap \pi_{Prereq.Condition}(\sigma_{NSSO}(Can-assign)) \neq \phi.$$

This means NSSO can assign role AP to Bob. The second step, based on Table 10,

$$Num = \pi_{APC}(\sigma_{RoleName=FPS}(ROLES)) = 0$$

It means no conflicts when assigning AP to Bob.

5.3 Application of the authorization revocation algorithm

Table 8 and Table 9 give the Can-revoke and a part of senior-junior relationship of the payment scheme.

Admin.role	Role Range
APSO	[AP, M1)
BankSO	[Bank, M2)
ShopSO	[Shop, M3)
NSSO	(FPS, DIR)
SSO	[FPS, DIR]

Table 8. Can-revoke of the payment scheme

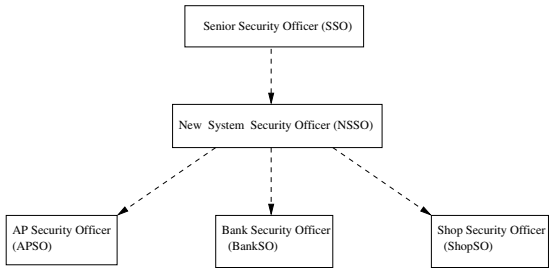

Figure 5. Administrative role assignment in the scheme

Suppose Bob is an explicit member of M1, QC, AU, AUDITOR, AP, FPS and E in the payment scheme. If Alice, with the activated administrative role APSO, weakly revokes Bob's membership from AP, the revocation is successful by the weak revocation algorithm Weak_revoke(APSO, R, AP),

$$R = \{M1, QC, AU, AUDITOR, AP, FPS, E\},$$

$$AP \in \pi_{RoleRange}(\sigma_{Admin.role=APSO}(Can-revoke)).$$

However, he continues to be a member of the senior roles to AP since both M1 and QC are senior roles to AP, therefore he can use the permission of AP. It is necessary to

Senior	Junior
FPS	E
OP	AP
QC	AP
M1	OP
M1	QC
M1	AP
Director	M1

Table 9. A part of SEN-JUN relation of the payment scheme

note that Alice should have enough power to weakly revoke Bob's membership from his explicitly assigned roles. For instance, if Alice has activated APSO and then tries to weakly revoke Bob's membership from M1, she is not allowed to proceed because APSO does not have the authority of weak revocation from M1 according to the can-revoke relation in Table 8.

Therefore, if Alice wants to revoke Bob's explicit membership as well as implicit membership from AP by weak revocation, she needs to activate SSO and weakly revoke Bob's membership from AP, QC and M1. This brings about the same result as strong revocation from AP by SSO. However, Alice does not need to revoke Bob's membership from FPS, AU, AUDITOR and E, because they are not senior roles to AP based on the role hierarchy of Figure 6.

For example, Bob is an explicit member of M1, QC, AU, AP, AUDITOR, FPS and E. If Alice with the activated administrative role SSO strongly revokes Bob's membership from AP, then he is removed not only from explicit membership in AP, also from explicit and implicit membership in all roles senior to AP. Using the strong revocation algorithm Strong_revoke(SSO, R, AP),

$$R = \{M1, QC, AU, AP, AUDITOR, FPS, E\}$$

$$R^* = \{M1, QC, OP, AU, AP, Bank, AUDITOR, FPS, E\}$$

Step 1, role AP is revoked from R since AP is an explicit member.
Step 2,

$$\begin{aligned} Sen &= R^* \cap \pi_{Senior}(\sigma_{Junior=AP}(SEN - JUN)) \\ &= \{M1, QC\}. \end{aligned}$$

This means Bob has been removed from M1, QC as well as AP after the strong revocation from AP. However, he still has a membership of FPS, AU, AUDITOR and E, since they are not senior roles to AP based on the role hierarchy of Figure 6. This brings out the same result as the one after weak revocation from AP, QC, M1 by SSO. Note that all implied revocations upward in the role hierarchy should be within the revocation range of the administrative roles that are active in a session. For instance, if Alice activates APSO and tries to strongly revoke Bob's membership from M1, she is not allowed to proceed because M1 is out of the APSO's can-revoke range in Table 8.

6. Related work

There are several other related works on role-based access control models [1], user-role assignment mechanisms developed for web-based intranets [16], Oracle system [2] and relational databases [11].

A role-based separation of duty language (RSL 99) has been recently proposed [1]. It has given a formal syntax and semantics for RSL99 and has demonstrated its soundness and completeness by using functions on conflicting role sets. The proposal is different from ours in two aspects. First, It does not consider the case of the management for conflicting roles. Therefore, there is no support to deal administrative roles with regular roles in the proposal. By contrast, our algorithms provide a rich variety of options that can deal the document of administrative roles with regular roles. Second, the algorithm RSL99 does not provide access control models. It only gives separation of duty (SOD) policies. By contrast, we present a number of specialized authorization algorithms for access control which allow administrators to authorize a role to users or revoke a role from users.

The interaction between RBAC and relational databases are presented in [11]. Two experiments are described. One is a role-based front end to a relational database with discretionary access control. Another one is a role graph to show the roles in a standard relational databases. Some relational concepts like roles, users and permissions etc are provided. Our model also support such concepts even though it has a large variety. However, the main difference between our algorithms and the scheme in [11] is, we focus on the solutions of the conflicts of roles and the latter focuses on the correlation of RBAC with discretionary access controls. Their work discusses the relationship between roles and discretionary access controls, they did not address how to allocate roles to users without conflicts. In our work, we developed detailed algorithms for allocating roles and checking the conflicts.

User-role assignment is also discussed in Oracle system by using SQL[2]. RBAC skills are partly used in [2]. Permissions can be assigned to users directly in Oracle system but not in our work. Another difference between our work and the work in Oracle system is in revocation process. We have demonstrated the weak revocation and the strong revocation that depend on different revocation requirements. However, the revocaton in [2] is as follows:

If you revoke a role to which other roles have been

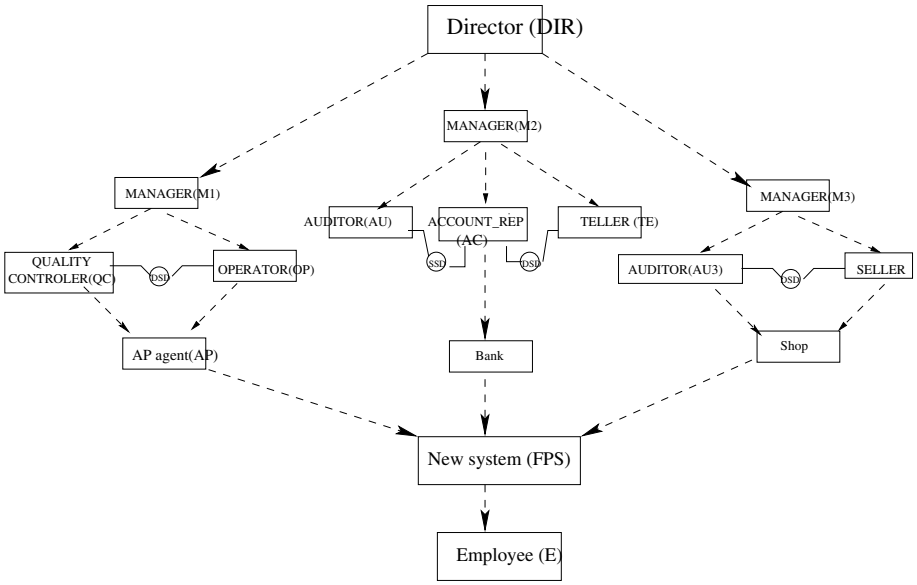

Figure 6. User-role assignment in the payment scheme

RoleName	EC	FPSC	APC	QCC	OPC	M1C	BaC	TEC	ACC	AUC	M2C	ShC	SAC	AU3C	M3C	DiC
E	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FPS	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
AP	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	-1
QC	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	-1
OP	0	0	0	-1	0	-1	-1	-1	-1	-1	-1	0	0	0	0	-1
M1	0	0	0	0	0	0	-1	-1	-1	-1	-1	0	0	0	0	-1
Bank	0	0	0	0	0	0	0	-1	-1	-1	-1	-1	0	0	0	-1
TE	0	0	0	0	0	0	0	0	0	-1	-1	-1	0	0	0	-1
AC	0	0	0	0	0	0	0	0	-1	0	-1	-1	0	0	0	-1
AU	0	0	0	0	0	0	0	0	-1	-1	0	-1	0	0	0	-1
M2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
Shop	0	0	0	0	0	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1
SELLER	0	0	0	0	0	0	-1	-1	-1	-1	-1	0	0	-1	-1	-1
AUDITOR	0	0	0	0	0	0	-1	-1	-1	-1	-1	0	-1	0	-1	-1
M3	0	0	0	0	0	0	-1	-1	-1	-1	-1	0	0	0	0	-1
Director	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10. ROLES in the payment scheme

granted, the entire set of privileges associated with every role will be revoked. Of course, if any of those roles and privileges had been granted directly to a user or a role affected by the revoke, they can still exercise the related privileges through the direct grant. page 302.

Finally, a user-role assignment for web-based intranet has been proposed in [16]. In such a model, the RBAC/web system with user-role assignment model (URA97) [15] is extended to decentralize the details of RBAC administration on the web. Authorizations can be given by administrators through prerequisite conditions. However, our work substantially differs from that proposal. Differences are due to the consistency problem arises in [16]:

It is very difficult to keep the consistency by reflecting security requirements between global network objects and local network objects if there are hundreds of roles and thousands of users in a system.

This problem is completely overcome in our algorithms because the algorithms focus on the conflicts between roles. The authorization granting algorithm is used to find conflicts and prevent some secret information to be derived while the strong revocation algorithm is used to check whether a role still has permissions of another role.

7. Conclusions

This paper has provided new authorization allocation algorithms for RBAC that are based on relational algebra operations. They are the authorization granting algorithm, weak revocation algorithm and strong revocation algorithm. The algorithms can automatically check conflicts when granting more than one role to a user in a system. They can prevent users from accessing unauthorized use of facilities when users change position within the organization and demand the modification of security rights. The roles can be allocated without compromising the security in RBAC and provide secure management for systems. The complexities of the algorithms are also analyzed. Furthermore, we have reviewed the consumer scalable anonymity payment scheme and discussed how to use the algorithms for the electronic payment scheme.

References

[1] Ahn G.J. and Sandhu R. The RSL99 Language for Role-Based Separation of Duty Constraints. In *4th ACM Workshop on Role-Based Access Control*, pages 43–54. Fairfax, VA, October, 1999.

[2] Austin D. and Lunawat V. etc. *Using Oracle 8*. QUE, A Division of Macmillan Computer Publishing, USA, 2000.

[3] Barkley J. F., Beznosov K. and Uppal J. Supporting relationships in access control using role based access control. In *Third ACM Workshop on RoleBased Access Control*, pages 55–65, October, 1999.

[4] Bertino E., Castano S., Ferrari E. and Mesiti M. . Specifying and enforcing access control policies for XML document sources. *World Wide Web, 3*, pages 139–151, Baltzer Science Publishers BV, 2000.

[5] David F.F., Dennis M.G. and Nickilyn L. An examination of federal and commercial access control policy needs. In *NIST NCSC National Computer Security Conference*, pages 107–116. Baltimore, MD, September, 1993.

[6] Feinstein H. L. Final report: Nist small business innovative research (sbir) grant: role based access control: phase 1. technical report. In *SETA Corp.*, Jan. 1995.

[7] Ferraiolo D. F. and Kuhn D. R. Role based access control. In *15th National Computer Security Conference*, pages 554–563, 1992.

[8] Ferraiolo D. F., Barkley J. F. and Kuhn D. R. Role-based access control model and reference implementation within a corporate intranet. In *TISSEC*, volume 2, pages 34–64, 1999.

[9] Goldschlag D., Reed M., and Syverson P. Onion routing for anonymous and private Internet connections. *Communications of the ACM*, 24(2):39–41, 1999.

[10] Okamoto T. An efficient divisible electronic cash scheme. In *Advances in Cryptology–Crypto'95*, volume 963 of *Lectures Notes in Computer Science*, pages 438–451. Springer-Verlag, 1995.

[11] Osborn S.L., Reid L.K. and Wesson G.J. On the Interaction Between Role-Based Access Control and Relational Databases. In *IFIP WG11.3 Tenth Annual Working Conference on Database Security*, pages 139–151, July, 1996.

[12] Rivest R. T. The MD5 message digest algorithm. *Internet RFC 1321*, April 1992.

[13] Sandhu R. Role-Based Access Control. *Advances in Computers*, 46, 1998.

[14] Sandhu R. Role activation hierarchies. In *Third ACM Workshop on RoleBased Access Control*, October, 1998.

[15] Sandhu R. and Bhamidipati V. "the ura97 model for role-based administration of user-role assignment". *T. Y. Lin and Xiao Qian, editors, Database Security XI: Status and Prospects*, North-Holland, 1997.

[16] Sandhu R. and Park J.S. Decentralized User-Role Assignment for Web-based Intranets. In *3th ACM Workshop on Role-Based Access Control*. Fairfax, Virginia, October, 1998.

[17] Wang H. and Zhang Y. Untraceable off-line electronic cash flow in e-commerce. In *Proceedings of the 24th Australian Computer Science Conference ACSC2001*, pages 191–198, GoldCoast, Australia, 2001. IEEE computer society.

[18] Wang H., Cao J. and Kambayashi Y. Building a consumer anonymity scalable payment protocol for the internet purchases. In *12th International Workshop on Research Issues on Data Engineering: Engineering E-Commerce/E-Business Systems*, San Jose, USA, Feb. 25-26, 2002.

[19] Wang H., Cao J. and Zhang Y. A consumer anonymity scalable payment scheme with role based access control. In *2nd International Conference on Web Information Systems Engineering*, pages 384–389, Kyoto, Japan, Dec. 3-6, 2001.

[20] Yiannis T. Fair off-line cash made easy. In *Advances in Cryptology–Asiacrypt'98*, volume 1346 of *Lectures Notes in Computer Science*, pages 240–252. Springer-Verlag, 1998.