Applications of the Oriented Permission Role-Based Access Control Model

Liang Chen
Information Security Group
Royal Holloway, University of London

Jason Crampton
Information Security Group
Royal Holloway, University of London

Abstract

Role-based access control and role hierarchies have been the subject of considerable research in recent years. In this paper, we consider three useful applications of a new role-based access control model that contains a novel approach to permissions and permission inheritance: one is to illustrate that the new model provides a simpler and more natural way to implement BLP model using role-based techniques; a second application is to make it possible to define separation of duty constraints on two roles that have a common senior role and for a user to be assigned to or activate the senior role; finally, we describe how a single hierarchy in new model supports the requirement of distinction between role activation and permission usage hierarchies. In short, the oriented permission model provides ways of implementing a number of useful features that have previously required ad hoc *and inelegant solutions.*

1 Introduction

Role-based access control (RBAC) has been the subject of considerable research in the last decade [5, 8, 12, 18] and is widely accepted as an alternative to traditional discretionary and mandatory access controls. Some important characteristics of RBAC that have led to its deployment in commercial computer systems and applications include policy neutrality, support for the principle of least privilege and ease of management.

A new role-based access control model [5] adopts a similar approach to RBAC96 [18] with respect to the role hierarchy and the user-role assignment relation, but proposes a new approach to permissions and permission inheritance within the role hierarchy. In this model, each permission is oriented and can be inherited in one of three ways within the hierarchy: by more senior roles, by less senior roles and by no other roles. Consequently, this model provides more flexibility than standard role-based models. Hereafter we refer to this model as OP-RBAC (Oriented Permission RBAC).

In this paper, we investigate various applications of OP-RBAC. Since the introduction of RBAC, several authors have discussed the relationship between RBAC and the Bell-LaPadula model (BLP) [13, 14, 15, 16]. Osborn *et al* [15] show that information flow policies in a number of different versions of BLP can be implemented in RBAC by the addition of a second role hierarchy and some constraints on the RBAC relations. However, these approaches are somewhat artificial and limited. The model for permission inheritance in OP-RBAC provides an alternative way of implementing BLP within the context of RBAC. We believe that this new approach is simpler, more natural, and more effective than existing work in this area.

Separation of duty has always been an important consideration in RBAC models. However, the standard RBAC model is not without its problems in this area. It is impossible for a user to be assigned to or activate a common senior role to two roles which are mutually exclusive in the role hierarchy. We will show how to use OP-RBAC to implement separation of duty constraints on two roles that have a common senior role and for a user to be assigned to or activate the senior role.

It has been shown that there are situations where it is useful to distinguish between role activation and permission usage inheritance [17]. Such a distinction has been made in both the ERBAC96 model [17] and the GTRBAC model [10], by introducing distinct role hierarchies. The final contribution of this paper is to prove that an instance of the ERBAC96 model can be transformed into an instance of the OP-RBAC model, which requires a single role hierarchy.

The rest of the paper is organized as follows. In the next section, we briefly review RBAC96, and formally present OP-RBAC and inter-relationships among the different components of the model. In Section 3, we consider three useful applications of OP-RBAC: one is to show how OP-RBAC can be used to implement BLP model with addition of a few constraints to the basic model. An second application is to illustrate that separation of duty requirements can be defined *and enforced* in a hierarchical RBAC model; the final one is to demonstrate how to implement the ERBAC96 model using OP-RBAC. We also discuss related work in these areas and compare them to our approaches. Section 4 concludes the paper with some suggestions for future work.

2 Background

In this section, we begin by providing an overview of RBAC96, and then go on formally define OP-RBAC.

2.1 RBAC96

We briefly summarize the RBAC96 model, on which OP-RBAC is based. We assume the existence of the following sets and relations:

- A partially ordered set of roles $RH \subseteq R \times R$. We will write $r \leqslant r'$ to denote that $(r, r') \in RH$ and $r \geqslant r'$ to denote that $(r', r) \in RH$. We write $\uparrow r$ to denote the set $\{r' \in R : r \leqslant r'\}$ and $\downarrow r$ to denote the set $\{r' \in R : r' \leqslant r\}$.

- A user-role assignment relation $UA \subseteq U \times R$.

  If $(u, r) \in UA$ then we say u is *explicitly* assigned to the role r. We denote the set of roles explicitly assigned to u by $R(u) = \{r \in R : (u, r) \in UA\}$ and all roles (explicitly and implicitly) assigned to u by $\downarrow R(u) = \{r' \in R : \exists r \in R(u), r' \leqslant r\}$.

- A set of sessions S.

  A user u activates a session $S(u)$ by selecting a subset of the roles to which u is assigned; that is $S(u) \subseteq \downarrow R(u)$.

- A set of permissions P.

- A permission-role assignment relation $PA \subseteq P \times R$.

  If $(p, r) \in PA$ then we say the permission p is *explicitly* assigned to the role r. We denote the set of roles to which p is explicitly assigned by $R(p) = \{r \in R : (p, r) \in PA\}$ and the set of roles authorized for p by $\uparrow R(p) = \{r' \in R : \exists r \in R(p), r \leqslant r'\}$.

- A request by a user u for permission p is granted if u has activated one of p's authorized roles, that is $S(u) \cap \uparrow R(p) \neq \emptyset$.

2.2 OP-RBAC

OP-RBAC is almost identical to RBAC96, but introduces a different approach to permission inheritance. As we shall see, this approach provides considerable more flexibility than standard role-based models. Formally, the model has the following characteristic features.

- A set of permissions P.

  Each permission is "oriented" with respect to inheritance and can be either "up", "down" or "neutral". That is, P is the disjoint union of P^+, P^- and P^0, where P^+ is the set of up permissions, P^- is the set of down permissions and P^0 is the set of neutral permissions.

  We denote the set of roles explicitly assigned to p by $R(p)$ and the set of roles authorized for p by the function $R_E : P \to 2^R$, where

$$R_E(p) = \begin{cases} \uparrow R(p), & \text{if } p \in P^+, \\ \downarrow R(p), & \text{if } p \in P^-, \\ R(p), & \text{if } p \in P^0. \end{cases}$$

  where $\uparrow R(p) = \{s \in R : \exists r \in R(p), r \leqslant s\}$ and $\downarrow R(p) = \{s \in R : \exists r \in R(p), r \geqslant s\}$. We say that $R_E(p)$ is the (set of) *effective* roles for p.

- A request by user u to invoke permission p is only granted if u has activated one of p's effective roles, that is $S(u) \cap R_E(p) \neq \emptyset$.

  This is obviously a generalization of the condition used in RBAC96, which treats all permissions as up permissions, and hence $R_E(p) = \uparrow R(p)$ for all p.

3 Applications of OP-RBAC

In this section, we demonstrate how, with the addition of a few constraints, OP-RBAC can be used to implement the BLP model. We also show how OP-RBAC can be used to remove some of the problems associated with the integration of role hierarchies and separation of duty requirements. We also investigate how to incorporate role activation and permission inheritance in a single OP-RBAC hierarchy. Finally, we give a comparison of our work with related work in the literature.

3.1 Implementing BLP using OP-RBAC

BLP [3] is probably the most widely known security model and implements an information flow policy designed to preserve the confidentiality of information. The key features of BLP are the partially ordered set of security labels L, the set of security functions Λ, the simple security property π^{ss}, the *-property π^* and the discretionary security property π^{ds}. Each subject (user) and object is associated with a security label which is determined by security function $\lambda(s)$ and $\lambda(o)$. π^{ss} requires that a subject s is authorized to `read` an object o if $\lambda(s) \geqslant \lambda(o)$. π^* introduces a current security label $\lambda_c(s)$, which enables a privileged user to downgrade his security level (where $\lambda_c(s) \leqslant \lambda(s)$), thereby allowing him to `append` to less privileged objects (provided $\lambda_c(s) \leqslant \lambda(o)$). As a consequence of π^{ss} and π^*, a subject s is authorized to `write` an object o if $\lambda(s) = \lambda_c(s) = \lambda(o)$. π^{ds} requires that all requests are also authorized by an appropriate entry in the protection matrix M.

In OP-RBAC, we interpret the user's security label in terms of his explicit user-role assignment(s) in the UA relation, and the user's current security label in terms of the roles he has chosen to activate in a session. In other words,

the user's security label is system-defined and the current security label is user-defined. This corresponds closely to $\lambda(s)$ and $\lambda_c(s)$ in BLP. Additionally, due to the uniqueness of security labels of subjects, we require that each user is assigned to a unique role, and can only run one session at a time and activate a single role. Formally, $R(u) = \{r\}$, and $S(u) = \{r'\}$ for some $r' \leqslant r$.

However, we can not regard the set of roles explicitly assigned to a permission, $R(p)$, as the security label of a permission, because permission usage in RBAC model is incompatible with BLP. In RBAC models, permission usage is based on an existential criterion; a user u can use permission p if *there exist* roles r and r' such that $(u, r) \in UA$, $(p, r') \in PA$ and $r' \leqslant r$. In BLP, permission usage is based on a universal criterion; a user u can use permission p if the security label of u, $R(u)$, dominates $R(p)$, the set of roles explicitly assigned to the permission. The incompatibility can be resolved by assigning each permission p to a unique role r. That is $R(p) = \{r\}$ for some $r \in R$ (as is assumed in existing approaches). Moreover, we require that all permissions for a particular object be assigned to a unique role r, thereby regarding role r as the security label of the object.

In order to satisfy π^{ss} and π^*, we must set some constrains for permissions and access requests checking in OP-RBAC. We require that if $p = (o, \texttt{read})$ then $p \in P^+$; if $p = (o, \texttt{append})$ then $p \in P^-$; if $p = (o, \texttt{write})$ then $p \in P^0$. We allow a `read` request if $R(u) \cap R_E(p) \neq \emptyset$; An `append` request is allowed if $S(u) \cap R_E(p) \neq \emptyset$; A `write` request is allowed if $R(u) = S(u) \cap R_E(p) \neq \emptyset$. For example, given $((o, \texttt{read}), r), ((o, \texttt{append}), r), ((o, \texttt{write}), r) \in PA$, any user u who is assigned to a role $r' \in \uparrow r$ is able to `read` the object o, and any user u who has activated a role $r' \in \downarrow r$ is able to `append` the object o. However, only the user u who is assigned to r and has activated r is able to `write` the object o.

Furthermore, it is at least possible to implement some coarse-grained discretionary properties using OP-RBAC. Suppose that we do not wish users with security label r to be able to `append` to objects with security label $r' > r$. We insist that $(o, \texttt{append})$ is a down permission. Instead the administrator can define this permission to be a neutral permission, so that only users with security label r' can `append` to o. Of course, the administrator can also assign this neutral permission to other roles $r'' \leqslant r'$ if desired. In other words, making certain permissions neutral rather than up or down, gives limited support for policies defined at the administrator's discretion.

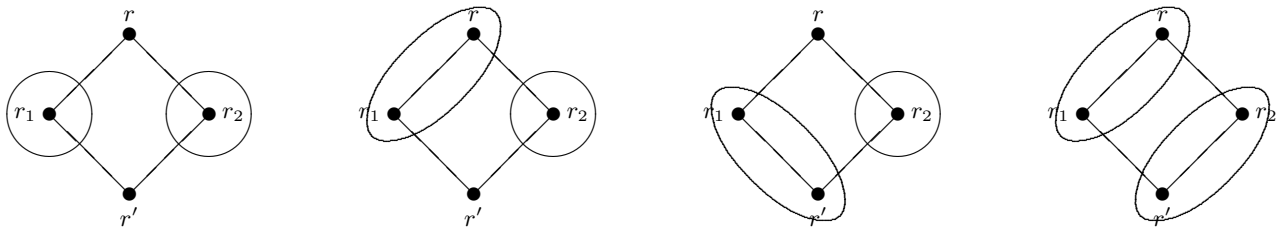
3.2 Separation of duty

Separation of duty is a widely recognized business principle that is used to prevent conflict of interests arising or to prevent fraudulent actions. At its simplest, it requires that if a sensitive task is comprised of two steps, then the same user can not perform both steps. Separation of duty in role-based systems has attracted considerable research interest in the literature [1, 4, 6, 11, 19].

If p and q are mutually exclusive permissions, the standard RBAC approach is to assign p and q to two different incomparable roles r_1 and r_2. *Static separation of duty* requires that no user can be assigned to both r_1 and r_2, whereas *dynamic separation of duty* requires that no user can activate both r_1 and r_2 in the same session. In standard hierarchical role-based systems, static separation of duty requires either that $R_E(p) \cap R_E(q) = \uparrow R(p) \cap \uparrow R(q) = \emptyset$, or for all roles $r \in \uparrow R(p) \cap \uparrow R(q)$, no user is assigned to or allowed to activate r. In the simplest case when p and q are assigned to different incomparable roles r_1 and r_2, either that r_1 and r_2 have no common senior role or that no user can be assigned to or activate any common senior role r. Unfortunately, this condition will usually completely remove the advantage that role hierarchies provide in reducing administration of the access control system. In short, separation of duty and role hierarchies are effectively mutually exclusive features of standard RBAC.

As in the standard RBAC approach, if p and q are mutually exclusive permissions and we want to ensure static separation of duty, we require in OP-RBAC that $R_E(p) \cap R_E(q) = \emptyset$ and for all $u \in U$, $R(u) \cap R_E(p) = \emptyset$ or $R(u) \cap R_E(q) = \emptyset$. Figure 1 illustrates the four ways of ensuring that for mutually exclusive permissions p and q assigned to roles r_1 and r_2 respectively, $R_E(p) \cap R_E(q) = \emptyset$. (The roles enclosed by a curve illustrate the effective set of roles for each permission.) The most direct way is to make p and q neutral permissions and assign them to roles r_1 and r_2 respectively, as show in Figure 1(a). Therefore, u can be assigned to more senior role r without acquiring the mutually exclusive permissions p and q. In addition, Figures 1(b)–1(d) shows that it is possible for u to be assigned to senior roles r or r' by defining p and q to be other types of permissions.

Note that the relationship between static separation of duty and dynamic separation of duty is different in OP-RBAC compared to standard RBAC. In standard RBAC, a user u is authorized for the set of permissions, $P(u) = \{p \in P : \exists r, r' \in R, (u, r) \in UA, (p, r') \in PA, r \geqslant r'\}$, and the set of permissions available to u is a session is monotonic with respect to the role hierarchy. In other words, if $r \leqslant r'$, it is always the case that the set of permissions available in a session in which r has been activated are a subset of those available in a session in which r' has been activated. Since the permissions available in a session are always a subset of those for which a user is authorized, we have that the enforcement of static separation of duty



(a) $p, q \in P^0$ (b) $p \in P^+, q \in P^0$ (c) $p \in P^-, q \in P^0$ (d) $p \in P^+, q \in P^-$

Figure 1: Implementing separation of duty using different types of permissions

implies the enforcement of dynamic separation of duty.

However, in OP-RBAC, the permissions available in a session are not monotonic with respect to the roles that have been activated. In other words, although we may have $r \leqslant r'$, it may not be the case that the set of permissions available in a session in which r has been activated are a subset of those available in a session in which r' has been activated. This is because permissions are oriented. In other words, if static separation of duty is enforced in OP-RBAC, it does not imply that dynamic separation of duty is satisfied in OP-RBAC. However, if we want to ensure dynamic separation of duty in OP-RBAC, we only require that for all $u \in U$, either $S(u) \cap R_E(p) = \emptyset$ or $S(u) \cap R_E(q) = \emptyset$ and $R_E(p) \cap R_E(q) = \emptyset$.

3.3 Usage and activation hierarchies

In most RBAC models, the role hierarchy serves two distinct purposes. A role is assumed to inherit the permissions assigned to roles below it in the hierarchy; this is called the *(permission) usage* aspect of role hierarchy. In addition, a user assigned to a particular role can also activate any subordinate roles in the hierarchy; this is called the *activation* aspect of role hierarchy.

Sandhu showed that making this distinction between usage and activation hierarchies has a number of useful applications [17]. In particular, it provides an alternative way of solving the incompatibility between static separation of duty constraints and the role hierarchy in RBAC96.

The ERBAC96 model (extended RBAC96) has a separate activation hierarchy, which extends the usage hierarchy of RBAC96 [17]. Formally, we have an usage hierarchy RH_u and an activation hierarchy RH_a, where $RH_u \subseteq RH_a$. In other words, $r \leqslant_u r'$ implies that $r \leqslant_a r'$. A user's interaction with the system is modelled by a session, where a user u activates a set of roles $S(u) \subseteq \downarrow_a R(u)$. The set of permissions for which u is authorized u in a session $S(u)$ is denoted by $P(s) = \bigcup_{r \in S(u)} \{y \in P : \exists r' \leqslant_u r, (y, r') \in PA\}$.[1]

Figure 2 on page 5 shows an example of an activation hierarchy and two different usage hierarchies, which will be used to demonstrate how to define the dynamic separation of duty requirement that a common senior role r_1 of two mutually exclusive roles r_2 and r_3 can be activated in ERBAC96. A user u, who is assigned to role r_1, is allowed to activate any junior roles in the activation hierarchy shown in Figure 2(a). However, if u activates role r_1, u can not acquire permissions assigned to roles r_2 and r_3, because there is no inheritance relation between roles r_1 and r_2 in the permission usage hierarchies shown in Figure 2(b) and 2(c).

We can implement this distinction between role activation and permission usage in OP-RBAC using only a single role hierarchy, up permissions and neutral permissions. Up permissions are inherited by more senior roles and neutral permissions are inherited by no other roles in the role hierarchy. The type of permissions is determined by the usage hierarchy, and the new permission assignment relation is determined by the usage hierarchy and the permission type. We can transform an ERBAC96 system (UA, PA, RH_u, RH_a, P) into a OP-RBAC system (UA, PA', RH_a, P') using the following procedure:

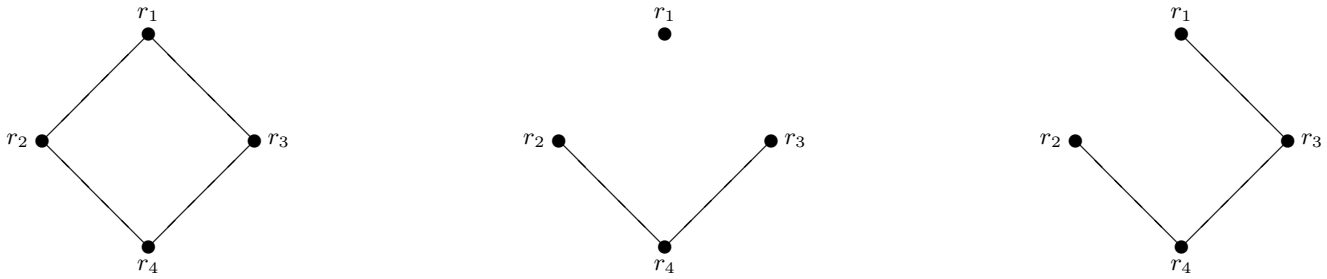
1. Let P^+ denote the set of up permissions and P^0 denote the set of neutral permissions;

2. Let PA^+ denote the permission-role assignments for up permissions and PA^0 denote the permission-role assignments for neutral permissions;

3. For all $r \in R$ such that $\uparrow_a r \neq \uparrow_u r$, and for all $p \in P$ such that $(p, r) \in PA$, we define $p \in P^0$, $(p, r) \in PA^0$, and for all $r' \in \uparrow_u r$, $(p, r') \in PA^0$;

4. For all $r \in R$ such that $\uparrow_a r = \uparrow_u r$, and for all $p \in P$ such that $(p, r) \in PA$, we define $p \in P^+$, $(p, r) \in PA^+$;

[1] GTRBAC [10] defines three different hierarchy types: permission-inheritance only hierarchy (I-hierarchy), activation only hierarchy (A-hierarchy) and inheritance-activation hierarchy (IA-hierarchy).



(a) $\leqslant_a$ (b) $\leqslant_u$ (c) $\leqslant_u$

Figure 2: Activation and usage hierarchies

r_1	p_1
r_2	p_2
r_3	p_2
r_3	p_3
r_4	p_4

(a) PA in ERBAC96

r_1	p_1	$+$
r_2	p_2	0
r_2	p_4	0
r_3	p_2	0
r_3	p_3	0
r_3	p_4	0
r_4	p_4	0

(b) PA' in OP-RBAC for Fig. 2(b)

r_1	p_1	$+$
r_1	p_2	0
r_2	p_2	0
r_3	p_2	0
r_3	p_3	$+$
r_4	p_4	$+$

(c) PA' in OP-RBAC for Fig. 2(c)

Figure 3: Transforming the ERBAC96 permission set and PA relation

5. For all $p \in P$ such that $(p, r) \in PA^0$ and $(p, r') \in PA^+$, we define $p \in P^0$, and for all $r'' \in \uparrow_u r'$, $(p, r'') \in PA^0$, and remove (p, r') from PA^+;

6. Define $P' = P^+ \cup P^0$ and $PA' = PA^+ \cup PA^0$.

We now show how the transformation works by taking the example of the ERBAC96 system illustrated in Figure 2 and the permission-role assignment relation in Figure 3(a). Firstly, we consider the example of usage hierarchy in Figure 2(b). Let us assume that the first role examined by the transformation procedure is role r_4. The first stage is to compute all roles which are senior to r_4 in the activation hierarchy, that is $\{r_1, r_2, r_3, r_4\}$ and all roles which are senior to r_4 in the usage hierarchy, that is $\{r_2, r_3, r_4\}$. Hence we find that $\uparrow_a r_4 \neq \uparrow_u r_4$; using Step 3 we define all permissions (only p_4 in our example) assigned to r_4 to be neutral permissions and assign all such permissions to r_2 and r_3. We continue to compute other roles (r_2, r_3, r_1). Finally, we output the set of neutral permissions $\{p_2, p_3, p_4\}$, the set of up permissions $\{p_1\}$ and the new permission-role assignment relation PA' shown in Figure 3(b).

For the second usage hierarchy in Figure 2(c), we firstly take the role r_4, for example, to be examined by the transformation procedure. We find that $\uparrow_a r_4 = \uparrow_u r_4$ and define p_4 assigned to r_4 to be an up permission (Step 4). After computing all roles (r_1, r_2, r_3, r_4), we find that $(p_2, r_2) \in PA^0$ and $(p_2, r_3) \in PA^+$ (Step 5). Hence, we add (p_2, r_3) and (p_2, r_1) to PA^0 and delete (p_2, r_3) from PA^+ (Step 5). Finally, the new permission role assignment relation PA' is generated as shown in Figure 3(c).

We now prove that the transformed OP-RBAC system is equivalent to the ERBAC96 system, in the sense that it returns the same answer as the original system for all possible access requests.

Theorem 1 *Let $\Sigma = (UA, PA, RH_a, RH_u, P)$ define an ERBAC96 system and $\Sigma' = (UA, PA', RH_a, P')$ define a OP-RBAC system derived from the ERBAC96 system in the manner described above. Then for all $p \in P$, $R_E(p)$ in Σ is equal to $R_E(p)$ in Σ'.*

Proof For convenience we write $R'_E(p)$ to denote $R_E(p)$ in Σ'.

We first prove that $R_E(p) \subseteq R'_E(p)$. Let $r \in R_E(p)$: then there exists r' such that $(p, r') \in PA$ and $r \geqslant_u r'$. There are two cases to consider. If $\uparrow_a r' \neq \uparrow_u r'$, then by Step 3, $(p, r) \in PA^0$ and $r \in R'_E(p)$. If $\uparrow_a r' = \uparrow_u r'$, then by Step 4, $(p, r') \in PA^+$. Since $r \geqslant_u r'$, by definition of ERBAC96, $r \geqslant_a r'$. Hence $r \in R'_E(p)$. (Note that if $(p, r') \in PA^0$ and $(p, r') \in PA^+$, then we add (p, r) to PA^0, using Step 5, and hence $r \in R'_E(p)$.) Therefore, $R_E(p) \subseteq R'_E(p)$.

We now prove that $R'_E(p) \subseteq R_E(p)$. Let $r \in R'_E(p)$: then there are two cases to consider. If $p \in P^0$ and $(p, r) \in PA^0$, then there exists $r' \leqslant_u r$ and $(p, r') \in PA$ (by Steps 3 and 5). By definition, $r \in R_E(p)$. Alternatively, if $p \in P^+$, then there exists $r \geqslant_a r'$ and $(p, r') \in PA^+$. By Step 4 $r \geqslant_u r'$ and $(p, r') \in PA$. Again, by definition, $r \in R_E(p)$. Therefore $R'_E(p) \subseteq R_E(p)$. The result now follows. ■

Corollary 2 *User u is authorized for p in Σ if and only if u is authorized for p in Σ'.*

Proof For any session s that user u can create in Σ, u can create exactly the same session in Σ', because the activation hierarchy RH_a is used in Σ'. u is authorized for p in session s, if and only if there exists $r \in s$ such that $r \in R_E(p)$. By Theorem 1, $r \in R_E(p)$ if and only if $r \in R'_E(p)$. ■

In summary, permission usage requirements in the system determine how to assign different types of permissions to roles in OP-RBAC. In certain situations, neutral permissions must be assigned to several hierarchical roles, which somewhat adds to the complexity of permission administration. On the other hand, the approach adopted in OP-RBAC offers simplicity by using a single role hierarchy. We might expect that it would be easier to administer an OP-RBAC system rather than an ERBAC96 one, for example. This would be an interesting direction for future work. In addition, we defined a transformation procedure that is for transforming an ERBAC96 system into a OP-RBAC system.

3.4 Related Work

There have been several attempts to implement BLP models using role-based models [13, 14, 15, 16]. Osborn *et al*'s approach [13, 14, 15] shows how the role-graph model can be configured to enforce information flow policies. In their approach the lattice of security label is defined separately and independently from the role graph. Each subject and object is assigned a security label as in BLP. Then the *r-level* of a role r, denoted by r-level(r), is defined to be the least upper bound of the security labels of the objects for which $(o, \texttt{read})$ is in the permissions of the role r; and the *a-level* of a role r, denoted by a-level(r), is the greatest lower bound of the security labels of the objects for which $(o, \texttt{append})$ is in the permissions of the role r. For all $(u, r) \in UA$, the security level of a user u must be greater than or equal to the r-level of r, and for all $(u, r) \in UA$, the security level of a user u must be less than or equal to the a-level of r. We think their approach for simulating the basic information flow policy using role-based access control is complicated, because their approach needs to introduce an extra lattice structure to determine the security labels of users and objects, and requires modification to the role-graph algorithms to compute the r-level and a-level of each role in the role graph model.

An alternative approach was developed by Sandhu *et al* [15, 16]. This involved defining two hierarchies, one for read roles and one for append roles. The append hierarchy is the dual of the read hierarchy: that is, $x \leqslant y$ in the append hierarchy if and only if $x \geqslant y$ in the read hierarchy. A number of constraints, similar to a number of ours were also defined. Each pair of permission $(o, \texttt{read})$ and $(o, \texttt{append})$ is assigned to exactly one matching pair of x_r and x_a roles in RH_r and RH_a respectively. Thereby, the security label of object o is implicitly defined to be x. Each session has exactly two matching roles y_r and y_a, and then y is regarded as the security label of user u. A read permission is granted if $y_r \geqslant x_r$ in RH_r and an append permission is granted if $y_a \geqslant x_a$ in RH_a. Note that RH_a is the dual of RH_r, thereby the condition of granting an append permission is $y_a \leqslant x_a$ in original lattice structure. We think it is not necessary to use a second hierarchy. More importantly, this approach can not cope with a compound (write) permission that has both read and append access rights to an object and does not consider discretionary aspects of the BLP model.

Compared with previous attempts, we believe OP-RBAC, with the addition of a few constraints, provides a more direct implementation of BLP model. We do not require an additional hierarchy and are able to support the assignment of "mixed" permissions (write permissions), which include both read and append access to objects. Perhaps the most significant contribution of the OP-RBAC model is the support for some limited discretionary policies, something existing work does not consider and would be ill-equipped to implement.

ERBAC96 [17] and GTRBAC [10] define multiple hierarchies that make it possible to implement separation of duties between two roles that have a common senior role. However, OP-RBAC provides much simpler way to solve the separation of duty requirement by assigning different types of permissions to these two roles in a single role hierarchy. In addition, it has been illustrated that usage and activation hierarchies in ERBAC96 can be implemented in a single role hierarchy in OP-RBAC. In other words, all the advantages of ERBAC96 are inherited by OP-RBAC.

4 Conclusions and future work

We have considered three useful applications of the OP-RBAC model, which arise because of its alternative treatment of permission inheritance. We noted that our approach provides a more natural implementation of BLP model using role-based techniques in a single role hierarchy. Our approach provides the first such implementation that supports the assignment of compound permissions

(both read and append access to objects) and demonstrates how it is possible to incorporate limited support for the discretionary security property of BLP, something that no existing work is able to do.

A second application is to make it possible for a user to be assigned to or activate a role when it is more senior than two mutually exclusive roles. To our knowledge, no other RBAC models are able to do this with a single role hierarchy.

Finally, we have described a way of supporting both permission usage and activation inheritance within a single hierarchy. We have defined a transformation that generates a OP-RBAC system that is equivalent to a given ERBAC96 system.

A first priority in future work is to develop a general multilevel secure model based on OP-RBAC with the consideration of inter-object information flow and complex permissions. We also like to investigate whether XACML is sufficient to define access control policies for the general model and hope to start work on the implementation of the model using Java. Other future work is to introduce OP-TRBAC (Oriented Permission Temporal RBAC) that takes into account of temporal aspects of OP-RBAC. Finally, we intend to extend existing delegation models [2, 7, 20] for RBAC to OP-RBAC. It has been suggested that a desirable feature in GTRBAC is to define "upward delegation", which allows a user to delegate a permission to roles more senior than the role to which the permission is assigned [9]. The flexible approach to permission inheritance in OP-RBAC, suggests that it would be straightforward to incorporate this feature in a delegation model for OP-RBAC.

References

[1] G.-J. Ahn and R. Sandhu. Role-based authorization constraints specification. *ACM Transactions on Information and System Security*, 3(4):207–226, 2000.

[2] E. Barka and R. Sandhu. A role-based delegation model and some extensions. In *National Information Systems Security Conference (NISSC)*, 2000.

[3] D. Bell and L. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report MTR-2997, Mitre Corporation, Bedford, Massachusetts, 1976.

[4] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1):65–104, 1999.

[5] J. Crampton. On permissions, inheritance and role hierarchies. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 85–92, 2003.

[6] J. Crampton. Specifying and enforcing constraints in role-based access control. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies*, pages 43–50, 2003.

[7] J. Crampton and H. Khambhammettu. Delegation in role-based access control. In *Proceedings of 11th European Symposium on Research in Computer Security*, pages 174–191, 2006.

[8] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.

[9] J. B. D. Joshi and E. Bertino. Fine-grained role-based delegation in presence of the hybrid role hierarchy. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 81–90, 2006.

[10] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role-based access control model. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):4–23, 2005.

[11] D. R. Kuhn. Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems. In *Proceedings of the second ACM workshop on Role-based access control*, pages 23–30, 1997.

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

[13] M. Nyanchama and S. L. Osborn. Modeling mandatory access control in role-based security systems. In *Proceedings of the Ninth Annual IFIP WG11 Working Conference on Database Security*, volume 51, pages 129–144, 1995.

[14] S. Osborn. Mandatory access control and role-based access control revisited. In *Proceedings of the 2nd ACM Workshop on Role-Based Access Control*, pages 31–40, 1997.

[15] S. Osborn, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security*, 3(2):85–106, 2000.

[16] R. Sandhu. Role hierarchies and constraints for lattice-based access controls. In *Proceedings of Fourth European Symposium on Research in Computer Security*, pages 65–79, 1996.

[17] R. Sandhu. Role activation hierarchies. In *Proceedings of the third ACM workshop on Role-based access control*, pages 33–40, 1998.

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

[19] R. Simon and M. Zurko. Separation of duty in role-based environments. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, pages 183–194, 1997.

[20] X. Zhang, S. Oh, and R. Sandhu. PBDM: A flexible delegation model in RBAC. In *Proceedings of 8th ACM Symposium on Access Control Models and Technologies*, pages 149–157, 2003.