A Critique of the ANSI Standard on Role Based Access Control

Ninghui Li **Ji-Won Byun** **Elisa Bertino**

CERIAS and Department of Computer Science
Purdue University
656 Oval Drive, West Lafayette, IN 47907-2086
{ninghui, byunj, bertino}@cs.purdue.edu

Abstract

The American National Standard Institute (ANSI) Standard on Role-Based Access Control (RBAC) was approved in 2004 to fulfil "a need among government and industry purchasers of information technology products for a consistent and uniform definition of role based access control (RBAC) features" [1]. The development of the ANSI RBAC standard represents an important milestone and will enhance portability and interoperability of applications and access control policies. The current version of the standard, however, has limitations, design flaws, and technical errors. In this article, we identify critical design problems in the current ANSI RBAC standard and suggest how they can be addressed. We also analyze several critical features of RBAC, such as sessions, hierarchies, and constraints, and discuss how they should be supported in RBAC models. We believe that our analysis will contribute to improvements in the RBAC standard and, more broadly, in the understanding of RBAC.

Keywords: Role-Based Access Control, Security, Authorization Management, Standards

1 Introduction

The past decade has seen an explosion of research in Role-Based Access Control (RBAC) [1, 2, 3, 4]. Hundreds of papers have been written on topics related to RBAC. The industry's interest in RBAC has also increased dramatically, with most major information technology vendors offering products that incorporate some form of RBAC. Today, all major Database Management System (DBMS) products support RBAC. In Windows Server 2003, Microsoft introduced Authorization Manager, which brings RBAC to the Windows operating systems. RBAC has also been used in Enterprise Security Management Systems such as IBM Tivoli Policy Manager [5] and SAM Jupiter [6].

Because of its relevance in products and applications for the management of enterprise security, RBAC has been the focus of intense standardization activities. The American National Standard Institute (ANSI) RBAC Standard was approved in 2004 to fulfil "a need among government and industry purchasers of information technology products for a consistent and uniform definition of role based access control (RBAC) features" [1]. The ANSI RBAC standard consists of two parts: a *Reference Model* and a *System and Administrative Functional Specification* (*Functional Specification* for short).The Reference Model defines sets of basic RBAC elements and relations, and the Functional Specification specifies the operations and functions an RBAC system should support. The RBAC standard includes four components: Core RBAC, Hierarchical RBAC, Static Separation of Duty (SSD) Relations, and Dynamic Separation of Duty (DSD) Relations. Both the Reference Model and the Functional Specification are divided into four parts corresponding to the four components.

The ANSI RBAC standard has gone through several rounds of open public review. An initial draft of the standard [7] was proposed at the 2000 ACM Workshop on RBAC. A panel was held at the ACM Workshop to discuss the document, and comments [8] have been published in the workshop proceedings. The second version, which addressed some of the issues raised by earlier critiques, appeared in ACM Transactions on Information and Systems Security (TISSEC) in 2001 [3] and was then submitted to the InterNational Committee for Information Technology Standards (INCITS) in October 2001. The final version was approved in February 2004 as the American National Standard ANSI INCITS 359-2004. Plans are underway to improve the standard and move the standard to ISO – International Organization for Standardization.

While the ANSI RBAC standard represents perhaps one of the most important milestones in RBAC research, it nonetheless has limitations, design flaws, and technical errors. Some of the most important issues with the standard that we discuss in this article include:

- The standard does not accommodate the design that only one role can be activated in a session, which we argue to be an appealing design that better supports the least privilege principle and the fail-safe defaults principle [9] and that has been used in some existing products.
- The Hierarchical RBAC component defines the inheritance relation to be a partial order, which we demonstrate to be inappropriate. Although using a partial order to represent a role hierarchy has been widely adopted in RBAC literature, it has a significant weakness when one considers updates to the role hierarchy. Clarifying this issue is key to the development of effective administrative models for RBAC.
- There are several possible interpretations of a role hierarchy, and these interpretations interact with each other and with other RBAC features in important ways. The standard fails to consider many of these interactions.
- There are a number of spelling and technical errors in the standard; these exist both in the 2001 TISSEC version [3] and in the 2004 ANSI version [1]. Due to the page limit, we point out only some of them in this article. A more complete list of the errors is reported in [10].

In this article, we elaborate on these issues and make suggestions for the improvement of the standard. The demonstrated importance of RBAC in research, product developments, and enterprise management dictates that solid and stable foundations must be established for RBAC. The intense past and current standardization efforts have achieved very important results in this respect. Our goal is to further improve the current state of the standard as well as the understanding of RBAC. Our discussions challenge some established assumptions in the RBAC research community. We believe that our discussions and findings will be of interest not only to standards bodies, product developers, and enterprise managers, but also to the research community. A sound standard model can be the basis upon which new research can be developed to further extend RBAC and can enhance technology transfer.

The remainder of this article is organized as follows. We provide a summary of the current ANSI RBAC Standard in Section 2. In Section 3 we discuss various issues in the current standard and make suggestions for improvement. We discuss related work in Section 4 and conclude with Section 5.

2 Overview of the ANSI RBAC Standard

In this section we provide the specifications of the four components in the ANSI RBAC standard.

Core RBAC is required in any RBAC system, and a particular RBAC system may optionally include any combination of role hierarchy, SSD, and DSD. For a more detailed description, the reader is directed to the standard [1, 3].

What is Role-Based Access Control (RBAC)?

Role-Based Access Control (RBAC) is an emerging paradigm for controlling access to computer resources. RBAC adds the notion of roles as a level of indirection between users and permissions. Roles are created based on job functions and/or qualifications of users. Permissions (i.e., privileges to access resources) are assigned to roles based on the requirements of job functions and/or the entitlement of qualifications. Users are made members of roles based on their job responsibilities and/or qualifications, thereby gaining permissions assigned to those roles. As such, in RBAC, users are granted permissions based on their roles, not on individual basis. This abstraction provided by roles significantly simplifies the management of permissions, and helps enforcing the principle of least privilege.

Most RBAC models include other features to further streamline the management of permissions and to help enforce other security principles. Two prominent features are role hierarchies and constraints. Roles may be organized into a hierarchy. Role hierarchies allow controlled sharing and aggregation of permissions among roles. For instance, permissions assigned to a junior role Doctor can be all shared by senior roles Cardiologist and Dermatologist, and permissions assigned to junior roles Programmer and Quality_Assurer can be aggregated for a senior role Project_Manager. Constraints can be used to enforce high-level security objectives such as the separation of duty principle or conflict of interest policy. For instance, constraints may prevent users from being assigned to two conflicting roles or activating them simultaneously.

Core RBAC The basic concept of RBAC is that permissions are assigned to roles and individual users obtain such permissions by being assigned to roles. Core RBAC captures this basic concept. The Core RBAC component in the Reference Model includes the following sets, functions and relations, which are taken verbatim from [1]. We use a footnote to point out a wording issue in them.

- $USERS$, $ROLES$, OPS, and OBS (users, roles, operations and objects respectively).
- $UA \subseteq USERS \times ROLES$, a many-to-many mapping user-to-role assignment relation.
- $assigned\_users : (r : ROLES) \rightarrow 2^{USERS}$, the mapping of role r onto a set of users. Formally: $assigned\_users(r) = \{u \in USERS \mid (u, r) \in UA\}$
- $PRMS = 2^{(OPS \times OBS)}$, the set of permissions.
- $PA \subseteq PRMS \times ROLES$, a many-to-many mapping permission-to-role assignment relation.
- $assigned\_permissions(r : ROLES) \rightarrow 2^{PRMS}$, the mapping of role r onto[1] a set of permissions. Formally: $assigned\_permission(r) = \{p \in PRMS \mid (p, r) \in PA\}$
- $Op(p : PRMS) \rightarrow \{op \subseteq OPS\}$, the permission to operation mapping, which gives the set of operations associated with permission p.
- $Ob(p : PRMS) \rightarrow \{ob \subseteq OBS\}$, the permission to object mapping, which gives the set of objects associated with permission p.
- $SESSIONS$ = the set of sessions
- $session\_users(s : SESSIONS) \rightarrow USERS$, the mapping of session s onto the corresponding user.

[1] As a reviewer pointed out, the use of the word "onto" here may give the wrong impression that the function $assigned\_permissions$ is an onto (i.e., surjective) function. It may be better to use "into", rather than "onto". The same applies to later occurrences of "onto".

- $session\_roles(s : SESSIONS) \rightarrow 2^{ROLES}$, the mapping of session s onto a set of roles. Formally: $session\_roles(s_i) \subseteq \{r \in ROLES \mid (session\_users(s_i), r) \in UA\}$
- $avail\_session\_perms(s : SESSIONS) \rightarrow 2^{PRMS}$, the permissions available to a user in a session = $\bigcup_{r \in session\_roles(s)} assigned\_permissions(r)$

Hierarchical RBAC The Hierarchical RBAC component introduces role hierarchies, which define an inheritance relation among roles in order to reduce the cost of administration. The Hierarchical RBAC component includes two types of role hierarchies: general role hierarchies and limited role hierarchies. Below are discussions and specifications for Hierarchical RBAC, taken verbatim from the standard [1]. We use footnotes to point out four errors in them.

> Role hierarchies define an inheritance relation among roles. Inheritance has been described in terms of permissions; i.e., r_1 inherits r_2 if all privileges of r_2 are also privileges of r_1. ... [2]
>
> **General Role Hierarchies**
>
> - $RH \subseteq ROLES \times ROLES$ is a partial order on $ROLES$ called the inheritance relation, written as $\succeq$, where $r_1 \succeq r_2$ only if all permissions of r_2 are also permissions of r_1, and all users of r_1 are also users of r_2, i.e., $r_1 \succeq r_2 \Rightarrow authorized\_permissions(r_2) \subseteq authorized\_permissions(r_1)$.
> - $authorized\_users(r : ROLES) \rightarrow 2^{USERS}$, the mapping of role r onto a set of users in the presence of a role hierarchy. Formally: $authorized\_users(r) = \{u \in USERS \mid r' \succeq r, (u, r') \in UA\}$
> - $authorized\_permissions(r : ROLES) \rightarrow 2^{PRMS}$, the mapping of role r onto a set of permissions in the presence of a role hierarchy. Formally: $authorized\_permissions(r) = \{p \in PRMS \mid r' \succeq r, (p, r') \in PA\}$ [3]
>
> Node r_1 is represented as an immediate descendant of r_2 by $r_1 \succ\succ r_2$, if $r_1 \succeq r_2$, but no role in the role hierarchy lies between r_1 and r_2. That is, there exists no role r_3 in the role hierarchy such that $r_1 \succeq r_3 \succeq r_2$, where $r_1 \neq r_2$ and $r_2 \neq r_3$. [4]
>
> **Limited Role Hierarchies**
>
> - General Role Hierarchies with the following limitation: $\forall r, r_1, r_2 \in ROLES, (r \succeq r_1 \ \wedge \ r \succeq r_2) \Rightarrow (r_1 = r_2)$. [5]

A limited role hierarchy forms a forest of inverted trees. In other words, there are a number of junior-most roles, and any of the other roles immediately dominates exactly one other role. As discussed in [7], an inverted tree facilitates sharing of resources (i.e., permissions). Resources made available to a junior-most

[2] This suggests that the role hierarchy is induced from the subset relation among the permissions that are assigned to these roles, which is incorrect. This statement implies that, if r_1 and r_2 are independently assigned the same permissions, then r_1 inherits r_2 and r_2 inherits r_1 at the same time, which is forbidden. This statement implies also that the role hierarchy may change when one changes the permission-to-role assignment relation, which is undesirable.

[3] $r' \succeq r$ should be $r \succeq r'$.

[4] The condition $r_1 \neq r_2$ should be $r_1 \neq r_3$.

[5] The definition is incorrect as it effectively limits the maximum height of role hierarchies to be two. To see this, observe that if $r \succeq r_1 \succeq r_2$, then the condition requires that $r_1 = r_2$. To correctly define the limitation, $(r \succeq r_1 \ \wedge \ r \succeq r_2)$ should be $(r \succ\succ r_1 \ \wedge \ r \succ\succ r_2)$.

role are also available to other more senior roles. However, an inverted tree does not allow aggregation of permissions from more than one role.

Constrained RBAC The Constrained RBAC component consists of two types of constraints: Static Separation of Duty (SSD) and Dynamic Separation of Duty (DSD). An SSD constraint is specified by a role set rs such that $|rs| \geq 2$ and a cardinality n such that $2 \leq n \leq |rs|$; it means that no user can be authorized for n or more roles in rs. Like SSD, a DSD constraint is specified by a role set rs such that $|rs| \geq 2$ and a cardinality n such that $2 \leq n \leq |rs|$; it means that no user may simultaneously activate n or more roles from rs in one session. The difference between SSD and DSD is that while an SSD constraint limits the roles for which a user can be authorized, a DSD constraint limits the roles that a user can activate in one session. The followings are taken verbatim from the standard.

Static Separation of Duty

- $\forall (rs, n) \in SSD, \forall t \subseteq rs : |t| \geq n \Rightarrow \bigcap_{r \in t} assigned\_users(r) = \emptyset$. [6]

Static Separation of Duty in the Presence of a Hierarchy

- $\forall (rs, n) \in SSD, \forall t \subseteq rs : |t| \geq n \Rightarrow \bigcap_{r \in t} authorized\_users(r) = \emptyset$.

Dynamic Separation of Duty

- $\forall rs \in 2^{ROLES}, n \in \mathbb{N}, (rs, n) \in DSD \Rightarrow n \geq 2, |rs| \geq n$, and
  $\forall s \in SESSIONS, \forall rs \in 2^{ROLES}, \forall role\_subset \in 2^{ROLES}, \forall n \in \mathbb{N}, (rs, n) \in DSD,$
  $role\_subset \subseteq rs, role\_subset \subseteq session\_roles(s) \Rightarrow |role\_subset| < n$. [7]

3 Issues in the ANSI RBAC Standard

In this section, we make five suggestions for improving the current RBAC standard. We explain our rationales underlying these suggestions by discussing the issues we have identified from the standard.

Suggestion 1 *The notion of sessions should be removed from Core RBAC and introduced in a separate component.*

The Core RBAC component includes the notion of sessions, where a session is defined as "a mapping between a user and an activated subset of roles that are assigned to the user" [1]. While we recognize that the notion of sessions is important for achieving least privilege in RBAC, we argue that there are important RBAC systems that do not need sessions and the standard should accommodate these systems. Therefore, we suggest that the notion of sessions not be included in Core RBAC, but be included in a new optional component.

While the notion of sessions is very useful in some applications (such as DBMSs), it is not applicable in some other applications. For example, in Enterprise Security Management (ESM) systems such as SAM Jupiter [6, 11] and IBM Tivoli [5], RBAC is used to provide the central management for authorizations over

[6] An alternative definition is $\forall (rs, n) \in SSD,\ \forall u \in USERS : |assigned\_roles(u) \cap rs| < n$.
[7] An alternative definition is $\forall (rs, n) \in DSD,\ \forall s \in SESSIONS : |session\_roles(s) \cap rs| < n$.

a number of heterogeneous target systems (e.g., operating systems, applications, and databases). In these ESM systems, users are assigned memberships in roles and gain permissions on abstract representations of the physical resources in the target systems. Then the ESM systems change the policy settings in target systems (e.g., via creating new accounts, changing group memberships of accounts, and changing access control lists) to provide users authorizations in the target systems. Users interact directly with the target systems to access resources; the ESM products use RBAC only to manage the policy settings in the target systems. The notion of sessions does not exist in such systems as permission usages happen in target systems and are outside the ESM systems.

The RBAC standard mandates: "Not all RBAC features are appropriate for all applications. As such, this standard provides a method of packaging features through the selection of functional components and feature options within a component, beginning with a core set of RBAC features that **must be** included in all packages." According to the above statement, ESM products such as SAM Jupiter and IBM Tivoli do not conform to the RBAC. This is undesirable, as these ESM products are generally considered to be among the most important applications of RBAC. The prospect of using these ESM products to greatly reduce administrative cost has been regarded as one of the strongest justifications for RBAC, and these products often drive the research on RBAC.

By including the notion of sessions in Core RBAC, thereby making it a mandatory feature, the current standard unnecessarily restricts RBAC. The basic concept of RBAC is that permissions are assigned to roles, and users obtain such permissions by being assigned to roles. This simple concept, with or without features such as sessions, has demonstrated to provide powerful and useful access control systems. Therefore, we argue that the notion of sessions should be included in a component other than Core RBAC. Our suggestion does not imply that the notion of sessions is not useful; it aims at making the standard more flexible and inclusive.

Suggestion 2 *The standard should accommodate RBAC systems that allow only one role to be activated in a session.*

Because of the way sessions are defined in the standard, an RBAC system that supports the feature of sessions must allow multiple roles to be activated in one session. However, in some RBAC systems (e.g., the one in [12], Informix according to [13], and Security Enhanced Linux [14]), only one role can be activated in a session. Let us consider the following two approaches:

1. **Single-Role Activation (SRA)**: Only one role can be activated in a session.
2. **Multi-Role Activation (MRA)**: Multiple roles can be activated in one session, and DSD constraints may be used to restrict concurrent activation of some roles.

Although the standard explicitly precludes SRA, we argue that SRA is sometimes more desirable.

It is easier to enforce the *least privilege* principle [9] with SRA than with MRA, for two reasons. First, with MRA, one needs an additional mechanism, namely DSD, to enforce the least privilege principle. For example, a user may be assigned both the Quality-Assurance role and the Developer role; however, the least privilege principle may mandate that the permissions assigned to the two roles should not all be available in one session. The SRA design automatically ensures that only one of these roles can be activated in any session. In MRA, this has to be achieved with DSD constraints. According to the *economy of mechanism* principle [9], SRA is a better design than MRA + DSD.

Second, to enforce the least privilege principle, one needs to review the sets of permissions that can be used in any session. With SRA, one just needs to examine the permissions assigned to each role independently, and does not need to consider how users are assigned to roles. With MRA, however, this is

more complicated, as independently examining each role is not sufficient. One has to consider all possible combinations of roles that can be activated together. The number of such combinations is likely much larger than the number of roles. Furthermore, enumerating all such combinations is more difficult, as one has to consider DSD constraints and the current user-to-role assignment.

A less obvious, but equally important, advantage SRA has over MRA is as follows. With SRA, if one wants to allow a user to be able to use permissions of several roles in one session, one has to define a new role that dominates all these roles and assign the user to this new role. On the other hand, with MRA, assigning a user to a role implicitly enables the user to activate this role together with all the other roles the user already has. When some of these combinations are undesirable, one has to explicitly specify DSD constraints to forbid undesirable combinations. Thus, a difference between SRA and MRA is that with SRA one has to do extra work to enable more accesses (by creating new roles) while with MRA one has to do extra work to forbid undesirable access (by adding constraints). As such SRA is consistent with the *fail-safe defaults* principle, which suggests that one should "base access decisions on permission rather than exclusion" [9], whereas MRA is not.

Finally, one may argue that using SRA one may end up with roles that have many permissions; however, if the system is implemented using MRA, then one ends up with users who are members of many roles and who effectively have all the permissions authorized for any role in the SRA case. In SRA, the aggregation of permissions is explicit, so that if there is one role with too much power, then this problem is quite obvious. In MRA, the aggregation of permission also exists, but is more stealthy and harder to detect.

Therefore, we argue that the standard should accommodate SRA so that a system that implements SRA rather than MRA can be considered to have implemented RBAC with sessions. The standard includes limited role hierarchies to accommodate systems that support only such role hierarchies; it should do the same for SRA. In fact, we would suggest that, in any RBAC implementation that needs to use sessions, one should carefully evaluate the tradeoff between SRA and MRA before choosing which one to adopt.

Suggestion 3 *The standard should make a clear distinction between base relations and derived relations.*

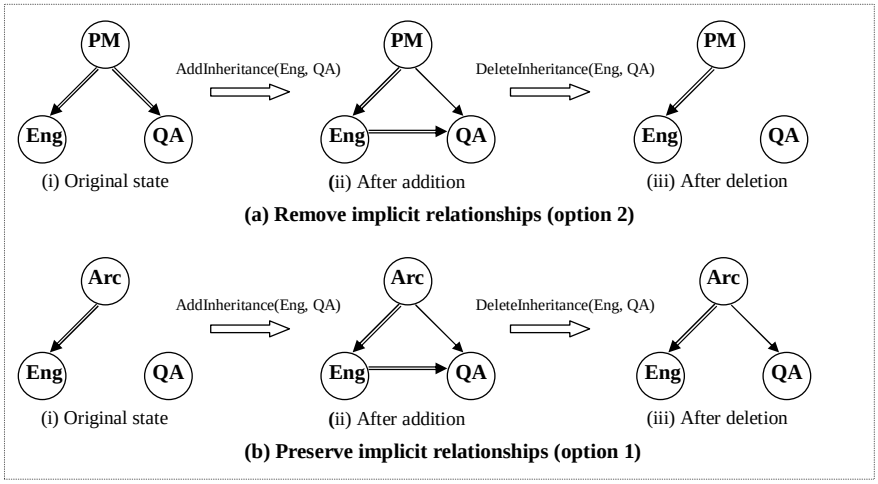
The standard does not clearly distinguish base relations and derived relations. For example, the Core RBAC specification includes both $UA \subseteq USERS \times ROLES$ and $assigned\_users : (r : ROLES) \rightarrow 2^{USERS}$. Furthermore, in the Functional Specification, invoking an administrative function (e.g., *AssignUser*, *DeassignUser* and *DeleteUser*) results in updates to both UA and $assigned\_users$. This indicates that UA and $assigned\_users$ are maintained independently. As each of UA and $assigned\_users$ can be derived from the other, having to explicitly maintain both relations unnecessarily complicates the specification of the administrative functions and gives rise to the possibility of inconsistencies.

We suggest that only one of UA and $assigned\_users$ be treated as a base relation and allowed to be updated by the administrative functions. The other one may be included as a derived, auxiliary function for the convenience of specifying other RBAC components. This distinction between base and derived relations should be explicitly made in the standard.

Other similar functions in Core RBAC include $assigned\_permissions$, which is derived from the permission assignment relation PA. The relations and functions in Core RBAC that deal with sessions includes $avail\_session\_perms$, which is derived from $session\_roles$ and PA.

Suggestion 4 *The Reference Model should maintain a relation that contains the role dominance relationships that have been explicitly added, and update this relation when the role hierarchy changes.*

In the Hierarchical RBAC component, a partial order relation RH is used to maintain the role hierarchy, and updates to the hierarchy are made to RH. While treating RH as a partial order has been the de facto



ProjManager (PM), Engineer (Eng), Quality Assurance (QA), and Architect (Arc)
$\Rightarrow$ and $\rightarrow$ represent an immediate predecessor relation and a non-immediate partial order relation, respectively.

Figure 1: Adding and deleting a role from RH

standard approach in RBAC literature (e.g., in the highly influential RBAC96 models [4]), we argue that this is inappropriate when updates to the role hierarchy are considered. We suggest that RH include only the role dominance relationships that have been explicitly added (RH is required to be *irreflexive* and *acyclic*), and that changes to the role hierarchy be carried out by changes to RH. The actual role hierarchy $\succeq$ is then defined to be the partial order entailed by RH, i.e., the reflexive and transitive closure of RH. For example, if the three role dominance relationships (r_1, r_2), (r_1, r_3), and (r_2, r_3) are added in that order, then the RBAC system should be able to distinguish this state from the state resulted from adding the two role dominance relationships (r_1, r_2) and (r_2, r_3), even though both result in the same partial order for role hierarchy. We now discuss the rationale for our suggestion.

In the standard, the role hierarchy is modeled using a partial order relation $\succeq$, and an immediate predecessor relation, denoted by $\succ\!\!\succ$, is defined as: $r_1 \succ\!\!\succ r_2$ if $r_1 \succeq r_2$ and there exists no other role r_3 such that $r_1 \succeq r_3 \succeq r_2$. Two functions $AddInheritance$ and $DeleteInheritance$ are defined such that only an inheritance relationship not in $\succeq$ can be added, only an inheritance relationship in $\succ\!\!\succ$ can be removed, and the role hierarchy resulted from removing such an inheritance relationship (r_1, r_2) is the reflexive and transitive closure of $\succ\!\!\succ$ with (r_1, r_2) removed.

Consider the RBAC state in Figure 1(a)(i), which includes the following role dominance relationships: $\mathsf{ProjManager} \succeq \mathsf{Engineer}$ and $\mathsf{ProjManager} \succeq \mathsf{QA}$. Suppose that when a product is about to be released, one wants engineers to also serve as QAs and adds a temporary relationship $\mathsf{Engineer} \succeq \mathsf{QA}$. This change results in the role hierarchy in Figure 1(a)(ii). After the release, one wants to delete the temporary relationship, expecting the hierarchy to return to the original state in Figure 1(a)(i). However, using $DeleteInheritance$ in the standard, the relationship $\mathsf{ProjManager} \succeq \mathsf{QA}$ will also be deleted, as after inserting relation $\mathsf{Engineer} \succeq \mathsf{QA}$, the $\succ\!\!\succ$ relation becomes $\{(\mathsf{ProjManager}, \mathsf{Engineer}), (\mathsf{Engineer}, \mathsf{QA})\}$. Thus, removing relation $\mathsf{Engineer} \succeq \mathsf{QA}$ results in the role hierarchy in Figure 1(a)(iii). Note also that in the role hierarchy in Figure 1(a)(ii), one cannot even remove the inheritance relation $(\mathsf{ProjManager}, \mathsf{QA})$ using $DeleteInheritance$, even though this was explicitly added previously, as $(\mathsf{ProjManager}, \mathsf{QA})$ is not in the immediate predecessor relation.

It has been suggested that one should keep all other role dominance relationships while removing one, e.g., in the administrative model for RBAC proposed in [15]. Using this interpretation, ProjManager $\succeq$ QA is maintained after deleting Engineer $\succeq$ QA. However, this introduces other problems. Consider the RBAC state in Figure 1(b)(i), which contains the following relationships: Architect $\succeq$ Engineer. After adding Engineer $\succeq$ QA, the state changes to Figure 1(b)(ii). After removing Engineer $\succeq$ QA, one would expect to return to the original state in Figure 1(b)(i). After all, the only reason that the Architect role dominates the QA role in Figure 1(b)(ii) is because one wants engineers to be able to serve as QAs and architects are (a kind of) engineers, and now one does not want engineers to be QAs anymore. However, the resulting state would be Figure 1(b)(iii), which is also undesirable.

In fact, the standard acknowledges that the two options exist and includes the following:

> When DeleteInheritance is invoked with two given roles, say Role A and Role B, the implementation system is required to do one of two things: (1) The system may preserve the implicit inheritance relationships that roles A and B have with other roles in the hierarchy. That is, if role A inherits other roles, say C and D, through role B, role A will maintain permissions for C and D after the relationship with role B is deleted; (2) A second option is to break those relationships because an inheritance relationship no longer exists between Role A and Role B. The question of which semantics the DeleteInheritance is left as an implementation issue and is not prescribed in this specification.

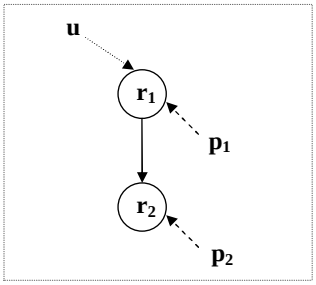
Observe that the above discussion is inconsistent with the definition of $DeleteInheritance$ in the standard, which adopts the second option. Furthermore, as previously discussed, neither option is satisfactory. As neither option is "more correct" than the other, one should not be forced to choose one or the other. The problem lies in the fact that, maintaining only a partial order, one cannot distinguish those role dominance relationships that have been explicitly added from those that are implied. For example, two different sets of role dominance relationships may entail exactly the same partial order. From the partial order, one cannot tell which set is the intended one. Maintaining only the derived partial order means that one does not maintain enough information about the current RBAC state and problems arise when changes to the role dominance relationships are made.

The solution we propose is to maintain explicitly added role dominance relationships in RH and use it to derive the implied partial order $\succeq$. For performance considerations, an RBAC system could choose to cache $\succeq$, as long as it can tell which dominance relationship was explicitly added and which was derived.

We emphasize that this issue should not be considered a minor implementation detail. Administration of RBAC is an open problem that is being actively researched [15, 16], and a consensus has yet to be reached. One key question, which has been overlooked so far, is how a role hierarchy should be maintained. When an RBAC paper mentions a role hierarchy, it almost always treats it as a partial order. This may be a reflection of the influence of earlier work on Mandatory Access Control (MAC) [17], where security levels are organized as a lattice (which is a partial order). As we argue above, the dynamic nature of role hierarchies (as opposed to the fixed security level lattices) requires a different approach.

Suggestion 5 *The semantics of role inheritance should be clearly specified and discussed.*

The RBAC standard implicitly allows three possible interpretations for a role hierarchy. For example, consider the situation illustrated in Figure 2: That $r_1 \succeq r_2$ may mean one or more of the following:



$UA = \{(u, r_1)\}$, $PA = \{(r_1, p_1), (r_2, p_2)\}$, $RH = \{r_1 \succeq r_2\}$

Figure 2: An RBAC state

1. **User Inheritance (**UI**)**: All users that are authorized for the role r_1 are also authorized for the role r_2. Under this interpretation, the user u is authorized for r_2 and is therefore authorized for the permission p_2. However, under this interpretation alone, r_1 is not authorized for p_2.

2. **Permission Inheritance (**PI**)**: The role r_1 is automatically authorized for all permissions for which r_2 is authorized. Under this interpretation alone, u is authorized for r_1 but not for r_2; however, u is authorized for p_2 as r_1 is authorized for p_2.

3. **Activation Inheritance (**AI**)**: When r_1 is activated in a session, r_2 is also activated in the session. Under this interpretation alone, u cannot activate r_2 directly (because u is not authorized for r_2 without UI); however, u can activate r_1, indirectly causing r_2 to be activated. In other words, u cannot use p_2 in a session without activating r_1.

All three kinds of inheritance semantics have been mentioned or alluded to in the standard. However, a clear specification and discussion of their relationships and interactions with other features in the standard are missing, and the standard is sometimes inconsistent about which semantics should be used by an RBAC system. We now provide a detailed analysis of the interactions among the three interpretations and other RBAC features, including SRA, MRA, SSD, and DSD.

a: At least one of UI, PI, AI should be used; otherwise, a role hierarchy is meaningless.

b: When there is no session, the notion of role activation does not exist; thus AI is not applicable. With SRA, only a single role can be activated in a session; AI cannot be used either.

c: When neither sessions nor constraints are used, UI and PI have exactly the same effect, as the only thing that matters in such systems is the set of permissions for which a user is authorized.

d: When there are (SRA or MRA) sessions, one can argue that PI (and, in the case of MRA, also AI) is more user friendly than having only UI, as u can use the role r_1 to have both p_1 and p_2 without knowing about the existence of r_2. In other words, the intricate details of how permissions are set up through roles can be partially hidden from users. With neither PI nor AI, the user u has to know r_2 and explicitly activate r_2 in order to use p_2.

e: When there are sessions, UI makes it easier to achieve the least privilege principle than using PI and AI, as a user can activate a less powerful role when that is sufficient for the current task.

	No Constraint	SSD	DSD	SSD + DSD
No Session	AI n/a; needs UI or PI [c]	AI n/a; needs UI [f]	n/a	n/a
SRA	AI n/a; needs PI [d]; needs UI [e]	AI n/a; needs PI [d]; needs UI [e,f];	n/a	n/a
MRA	needs PI or AI [d]; needs UI [e]	needs PI or AI [d]; needs UI [e,f]	needs PI or AI [d]; needs UI [e]; needs AI [g]	needs PI or AI [d]; needs UI [e,f]; needs AI [g]

Figure 3: Summary of our analysis. For example, the phrase "needs UI [f]" in the cell in the "No Session" row and SSD column means that it is desirable to have UI in this case, and the justification is given under bullet f in the text.

f: When there are SSD constraints, if one uses just PI and not UI, the intention of SSD constraints can be circumvented. For example, if two roles r_1 and r_2 are declared to be mutually exclusive, the intention is that no user should be authorized for the combined permissions of r_1 and r_2. However, with just PI and not UI, one can define a role r_3 to dominate both r_1 and r_2 and assign a user u to r_3 without violating the constraint, as u is not authorized for r_1 or r_2 without UI. Thus, the absence of UI results in less strict SSD constraints which apply to users directly assigned to the conflicting roles, but not to users assigned to their senior roles.

g: DSD constraints only make sense when MRA sessions exist. With DSD constraints, having PI but not AI lessens the strictness of constraints for the reason similar to the above. For example, suppose that $r_1 \succeq r2$ and $r_3 \succeq r_4$ and that r_2 and r_4 are declared to be dynamically mutually exclusive. With PI but not AI, a user can exercise the combined permissions from both r_2 and r_4 without violating the constraint, as the user can use the permissions of r_2 and r_4 without activating them. Therefore, the absence of AI leads to less strict DSD constraints which are enforced only to users who are directly assigned to the conflicting roles.

We summarize our analysis in Figure 3. The standard is unclear on which of the three interpretations should be used. In Section A.2.2, the standard reads "When that given role is activated by a user, the question of whether the inherited roles are automatically activated or must be explicitly activated by a user is left as an implementation issue and no one course of action is prescribed as part of this specification." However, from the ways functions such as $AddActiveRole$ are defined, one can infer that the Functional Specification adopts the approach of UI, PI, but no AI. The $AddActiveRole$ function adds only the role that has been explicitly specified to the $session\_roles$ relation, and the check for DSD constraints checks only the roles in $session\_roles$. As discussed above, this means that the effect of DSD constraints can be circumvented, which may be undesirable. Our suggestion is to specify and discuss the three interpretations for role hierarchies, to make recommendations about which interpretations should be used, and to define the Functional Specification in a way that is consistent with the recommendations. One possible approach that is consistent with our analysis is to use both UI and PI at all times and add AI whenever MRA is used.

4 Related Work

The notion of roles was first introduced to access control in the context of database security [18, 12] as a means to group permissions together to ease security administration. The term "Role Based Access Control"

was first coined by Ferraiolo et al. [2, 19]. Sandhu et al. [4] developed the influential RBAC96 family of RBAC models. Sandhu [20] explained the rationale under the RBAC96 models, e.g., why sessions were considered a fundamental concept in RBAC96. Based on these earlier works on RBAC, the efforts for the NIST RBAC standard emerged. The first proposal appeared at the 2000 ACM Workshop on RBAC [7]. It is organized into four levels of increasing capabilities. Flat RBAC requires the essential elements that capture the basic concept of RBAC. Hierarchical RBAC requires supporting role hierarchies. Constrained RBAC adds a requirement for enforcing separation of duties (SOD), which includes static SOD (based on user-role assignment) and dynamic SOD (based on role activation). Lastly, Symmetric RBAC adds a requirement for permission-role review. In [8], Jaeger and Tidswell question the usefulness of the proposed RBAC unified reference model, partly because many issues were left as implementation decisions and the draft standard does not provide any guidelines. They argue that except the first level, the levels are orthogonal extensions and that many of the concepts in the proposed model are either vaguely specified or unnecessarily restrictive. They also argue that administrative features of RBAC should be put in the standard. The standard adopted the suggestion in [21] to make it possible to pick and choose features such as role hierarchies and constraints. It did not adopt the suggestion to standardize administrative features in RBAC. This appears to be a correct decision as administration of RBAC remains an active research area and a consensus has not been reached in the community.

Sandhu [22] discussed the permission-usage aspect of role hierarchies, which corresponds to our PI interpretation, and the role-activation aspect of role hierarchies, which corresponds to UI, and suggested having two hierarchies, one for each. Having multiple hierarchies increases the flexibility at the cost of complicating the RBAC model. As the standard uses a single hierarchy, we focus our discussions on a single hierarchy. Moffett [23] examined the relationship between the inheritance properties of role hierarchies and control principles such as separation of duties, delegation and supervision, and identified three different types of role hierarchies.

5 Conclusions

RBAC represents an important milestone in research and development in access control models and techniques. Its wide-scale adoption in products, such as database management systems and enterprise management systems, has shown its potential from a commercial point of view. Current trends, like pervasive and mobile computing as well as integration of heterogeneous systems and enterprises, call for interoperability among access control policies. In this respect, the fact that an initial standardization for RBAC has been developed is important. However, due to the complexity of RBAC, several aspects of the standard require critical examination and careful revisions. In this article, we have reported the results of a detailed analysis we have carried out on the current version of the ANSI RBAC standard [1, 3], aiming to identify critical problems of the standard and possible solutions to these problems.

Specifically, we have pointed out a number of technical errors and limitations in the ANSI RBAC standard and suggested how they can be addressed. As a key contribution, we have shown that to maintain a role hierarchy, one should maintain the role dominance relationships that have been explicitly added and distinguish them from the derived relationships. This challenges a basic assumption made by existing administrative models for RBAC [15, 16], and leads to new approaches to administrative models for RBAC. As another key contribution, we have clarified three interpretations of role hierarchy: user inheritance, permission inheritance and activation inheritance. We also discussed their relative benefits and limitations, especially in their interaction with other RBAC features such as sessions and constraints. We hope that our

analysis will contribute to improvements in the RBAC standard and, more broadly, in the understanding of RBAC.

References

[1] ANSI. American national standard for information technology – role based access control. ANSI INCITS 359-2004, February 2004.

[2] David F. Ferraiolo and D. Richard Kuhn. Role-based access control. In *Proceedings of the 15th National Information Systems Security Conference*, 1992.

[3] David F. Ferraiolo, Ravi S. Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and Systems Security*, 4(3):224–274, August 2001.

[4] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[5] Gunter Karjoth. Access control with IBM Tivoli Access Manager. *ACM Transactions on Information and System Security*, 6(2):232–257, May 2003.

[6] Roland Awischus. Role based access control with the security administration manager (SAM). In *Proceedings of the second ACM workshop on Role-based access control table of contents (RBAC '97)*, pages 61–68, 1997.

[7] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The NIST model for role-based access control: towards a unified standard. In *Proceedings of the Fifth ACM Workshop on Role-Based Access Control (RBAC 2000)*, pages 47–63, 2000.

[8] Trent Jaeger and Jonathon E. Tidswell. Rebuttal to the NIST RBAC model proposal. In *Proceedings of the Fifth ACM workshop on Role-Based Access Control (RBAC 2000)*, pages 65–66, 2000.

[9] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[10] Ninghui Li, Jiwon Byun, and Elisa Bertino. A critique of the ANSI standard on role based access control. Technical Report TR 2005-29, Purdue University, 2005.

[11] Axel Kern. Advanced features for enterprise-wide role-based access control. In *Proceedings of the 18th Annual Computer Security Applications Conference*, pages 333–343, December 2002.

[12] Robert W. Baldwin. Naming and grouping privileges to simplify security management in large databases. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 116–132, May 1990.

[13] Chandramouli Ramaswamy and Ravi Sandhu. Role based access control features in commercial database management systems. In *Proceedings of the 21st National Information Systems Security Conference*, Oct 1998.

[14] NSA. Security enhanced linux. http://www.nsa.gov/selinux/.

[15] Jason Crampton and George Loizou. Administrative scope: A foundation for role-based administrative models. *ACM Transactions on Information and System Security*, 6(2):201–231, May 2003.

[16] Ravi S. Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based aministration of roles. *ACM Transactions on Information and Systems Security*, 2(1):105–135, February 1999.

[17] D. Elliott Bell and Leonard J. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, Mitre Corporation, March 1976.

[18] T. C. Ting. A user-role based data security approach. In C. Landwehr, editor, *Database Security: Status and Prospects. Results of the IFIP WG 11.3 Initial Meeting*, pages 187–208. North-Holland, 1988.

[19] David F. Ferraiolo, Janet A. Cuigini, and D. Richard Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of the 11th Annual Computer Security Applications Conference (ACSAC'95)*, December 1995.

[20] Ravi S. Sandhu. Rationale for the RBAC96 family of access control models. In *Proceedings of the First ACM Workshop on Role-Based Access Control*, 1996.

[21] Trent Jaeger and Jonathon E. Tidswell. Practical safety in flexible access control models. *ACM Transactions on Information and System Security*, 4(2):158–190, May 2001.

[22] Ravi S. Sandhu. Role activation hierarchies. In *Proceedings of the Third ACM Workshop on Role-Based Access Control (RBAC 1998)*, pages 33–40, October 1998.

[23] Jonathan D. Moffett. Control principles and role hierarchies. In *Proceedings of the Third ACM Workshop on Role-Based Access Control (RBAC 1998)*, October 1998.