Analyzing and Developing Role-Based Access Control Models

by

Liang Chen

A thesis submitted in fulfilment of the requirements for the Degree of

Doctor of Philosophy

in the

University of London

Information Security Group

Department of Mathematics

Royal Holloway, University of London

2011

Declaration

These doctoral studies were conducted under the supervision of Dr Jason Crampton.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Department of Mathematics, Royal Holloway, University of London as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Liang Chen

November, 2010

Abstract

Role-based access control (RBAC) has become today's dominant access control model, and many of its theoretical and practical aspects are well understood. However, certain aspects of more advanced RBAC models, such as the relationship between permission usage and role activation and the interaction between inheritance and constraints, remain poorly understood. Moreover, the computational complexity of some important problems in RBAC remains unknown. In this thesis we consider these issues, develop new RBAC models and answer a number of these questions.

We develop an extended RBAC model that proposes an alternative way to distinguish between activation and usage hierarchies. Our extended RBAC model has well-defined semantics, derived from a graph-based interpretation of RBAC state.

Pervasive computing environments have created a requirement for access control systems in which authorization is dependent on spatio-temporal constraints. We develop a family of simple, expressive and flexible spatio-temporal RBAC models, and extend these models to include activation and usage hierarchies. Unlike existing work, our models address the interaction between spatio-temporal constraints and inheritance in RBAC, and are consistent and compatible with the ANSI RBAC standard.

A number of interesting problems have been defined and studied in the context of RBAC recently. We explore some variations on the set cover problem and use these variations to establish the computational complexity of these problems. Most importantly, we prove that the minimal cover problem – a generalization of the set cover problem – is NP-hard. The minimal cover problem is then used to determine the complexity of the inter-domain role mapping problem and the user authorization query problem in RBAC. We also design a number of efficient heuristic algorithms to answer the minimal cover problem, and conduct experiments to evaluate the quality of these algorithms.

Contents

List of Figures

List of Tables

Acknowledgements

Firstly, I would like to express my gratitude to my supervisor, Dr Jason Crampton, for giving me the opportunity to study for a PhD. His excellent guidance, critical feedback and active involvement have made a huge contribution to the quality and presentation of my work. I also thank Jason for his endless patience and constant encouragement whenever I was in doubt in the last years. The experience to work with Jason has been invaluable, and this experience will be most useful and important for my future.

I am grateful to my advisor, Professor Chris Mitchell, for his constructive comments on my work in the annual review meetings. I am also thankful to Chris for granting me a College Research Studentship, which was extremely helpful in alleviating the financial burden of my PhD studies.

I owe a great debt to colleagues within the Information Security Group, Royal Holloway College: Dr Jiqiang Lu, Dr Hoon Wei Lim, Dr Geong Sen Poh, Chunhua Chen, and Ziyad Al-Salloum for their inspiring discussions of PhD research, and generally being great office mates. In particular, I would like to thank Hemanth Khambhammettu for the many stimulating conversions we have had, and for the time he spent reading and commenting on the chapters of the thesis. I also would like to thank Dr Charles Morisset for proof reading of chapters of the thesis despite his busy schedule.

During my stay in England, I have made many new friends in the last few years but two of these in particular are considered to be my closest and best friends: Yichuan Chen and Robert Williams. I feel very fortunate to have met Yichuan as I can share my most inner thoughts with him, and he has been of great help to me in so many different ways. I am also very grateful to Robert for his generous help and kindness with any problems I have encountered in my everyday life.

Finally, I would like to thank my parents for their continuous support and encouragement. I am especially indebted to my fiancee, Zheng Ning, for her unconditional love and support, who also makes my life better and complete in the last three years.

Chapter 1

Introduction

The protection of information in multi-user computer systems has become increasingly important as a result of the rapid development and widespread deployment of computer systems in our daily life. The most common protection measures used in computer system are *prevention*, *detection* and *recovery* [46]. Prevention is applied to prevent information from being damaged. In other words, its purpose is to prevent all unauthorized access to information. Detection allows us to detect when information has been damaged, how it has been damaged, and who has caused the damage. Recovery allows us to restore the information that has been damaged or to assess and repair any damage to the information. In this thesis, we are only concerned with prevention.

The prevention measure is regarded as the traditional core of computer security, which usually attempts to achieve three security goals: *confidentiality*, *integrity* and *availability*. These security goals are concerned with prevention of unauthorized disclosure of information, unauthorized modification of information, and unauthorized withholding of information, respectively [26]. In this thesis, we focus on studying *access control*, one of the most important prevention techniques in computer systems. In particular, it can help enforce confidentiality and integrity, and can also provide a basis

for availability. For example, an attacker who gains unauthorized access to a sensitive document is likely to have little trouble making this document unavailable to valid and authorized users. Generally speaking, access control is concerned with limiting access by users to protected resources. In multi-user computer systems, some resources may be publicly accessible by all users, some may only be accessible by a restricted audience, and some may be private to the user who creates those resources. These requirements are usually expressed by *access control policies* that generally specify who is allowed to access which resources in a computer system. *Access control mechanisms* are used to implement access control policies, and ensure that users' requests to access resources are only granted if those requests are authorized by the policies.

Access control has been the subject of considerable research and developments over the last 30 years. The Trusted Computer System Evaluation Criteria (TCSEC), commonly known as the "Orange Book" [89], defines two different types of access control: *discretionary access control* and *mandatory access control*. Discretionary access control restricts access to resources based on the identity of users. The two key features of discretionary access control is that each resource is required to have an *owner*, and the owner is able to control who can access the resources she owns. Discretionary access control is the most widely used form of access control being used in operating systems such as UNIX and Windows NT. Operating systems usually implement discretionary access control on the basis of *access control lists*. That is, each resource in the system is associated with a list of users who are entitled to access that resource. A request from a user to access a given resource is granted if the user is in the access control list.

Mandatory access control constrains the interactions of users with resources on the basis of security attributes associated with users and resources. A request from a user to access a resource is granted if certain inequalities comparing the security attributes of the user and the resource are satisfied. Unlike discretionary access control,

changes to the authorization policies in mandatory access control are controlled by a higher authority, rather than individual users. In other words, system administrators define mandatory access control policies that can be centrally enforced for all users. The Bell-LaPadula model [12] is probably the most widely known security model and incorporates a mandatory information flow policy for confidentiality [35]. Informally, the mandatory information flow policy comprises the *simple security property* and the **-property*. The simple security property requires that a user is only allowed to *read* a resource if the security attribute of the user is at least as high as that of resource being read. Conversely, the *-property requires that a user is only allowed to *write* to a resource if the security attribute of the resource is at least as high as that of the user. These kinds of mandatory access control policies are commonly used in military applications and commercially sensitive applications.

However, modern business information systems typically comprise millions of resources and thousands of users. The complexity of such systems makes it difficult to employ discretionary access control that might require the management of millions of access control lists or other similar access control data structures. Conversely, mandatory access control has been found to be too restrictive for general organizational information systems [39].

Role-based access control (RBAC) [40, 70, 83] has emerged as the primary alternative to discretionary access control and mandatory access control, because of its potential to reduce the complexity and cost of access control administration in organizational systems. The basic idea of RBAC is to introduce the concept of a *role*, which acts as a "bridge" between users and permissions[1]. A user can use a permission by activating a role to which the user and the permission are assigned. This approach

[1]Informally, the term permission refers to some combination of resource and action that users attempts to perform.

of associating users and permissions with roles greatly simplifies the management of permissions for which a user is authorized. For example, if an employee is required to change his position within an organization, we simply assign a new set of roles to this user without having to directly assign a completely new set of permissions to him.

RBAC further reduces the administrative burden by introducing the idea of a *role hierarchy*. The role hierarchy generally supports two different types of inheritance: permissions are inherited upwards and the set of roles available to a user is aggregated downwards. For example, if role r is senior to r' in the role hierarchy, then any permission assigned to r' is implicitly assigned to r, and any user assigned to r can activate r'. These two types of inheritance are called *permission usage* and *role activation* respectively. In other words, a role hierarchy can be used to reduce the number of explicit assignments in the user-role and permission-role relations.

The fundamental concepts of RBAC are now well established and are detailed in the recent release of the ANSI RBAC standard [2]. RBAC has also been incorporated into several commercial software products offered by major information technology vendors, such as Authorization Manager in Windows Server 2003, Oracle Access Manager, IBM Tivoli Policy Manager [59] and SAM Jupiter [6]. Although RBAC has obtained considerable maturity in many theoretical and practical aspects, the uses of role hierarchies in RBAC has a number of inconvenient consequences which have not been thoroughly addressed. In addition, certain aspects of more advanced RBAC models, such as the relationship between permissions usage and role activation and the interaction between inheritance and constraints, remain poorly understood. Moreover, the computational complexity of some important problems in RBAC remains unknown. The main goal of this thesis is to examine issues noted above, to develop new RBAC models for addressing some of these issues, and to analyze and answer a number of outstanding problems in RBAC.

1.1 Motivation

At the most general level, this thesis is concerned with role hierarchies and the way in which they interact with other parts of the RBAC model. There is a substantial body of work in the literature that introduces new hierarchies or constraints, without ever properly considering the combined effect of these new features or their effect on the authorization semantics of the resulting models. In this section, we present some of these problems which provide the motivation for our research in this thesis.

1.1.1 The role hierarchy and inheritance

Role-based access control is one of the most powerful access control paradigms with many potential applications [39]. An important characteristic that makes RBAC so attractive is that it can be configured to support a wide variety of access control policies, including traditional discretionary and mandatory access control policies [39], as well as organization-specific policies, such as separation of duty policies [39]. However, it has been observed that the inheritance semantics of the role hierarchy makes it awkward for RBAC to implement the Bell-LaPadula model and dynamic separation of duty constraints [29]. We now explain these issues in more detail, and describe how existing work provides limited solutions to address some of them.

The Bell-LaPadula model

The inheritance of permissions within a role hierarchy is always upwards, that is, if a permission p is assigned to a role r, any user assigned to a role at least as senior as r can exercise permission p. This is analogous to the use of read permissions in the Bell-LaPadula model, where a permission of read access to a resource is available to any user whose security label is at least as high as that of the resource. However,

the *-property in the Bell-LaPadula model usually requires that write permissions are inherited downwards, which can not be directly handled in the standard RBAC model.

There have been several attempts to simulate the Bell-LaPadula model using role-based approaches [29, 69, 72, 73, 81, 82]. These previous attempts typically require changes or additions to the underlying RBAC model, such as the inclusion of a second role hierarchy [73, 81, 82] and the addition of an external data structure for determining security attributes of users and resources [29, 69, 72]. However, we are unaware of any attempt to simulate the behaviour of the Bell-LaPadula model using a single role hierarchy. In particular, it is not possible to support the assignment of "mixed" permissions in existing work, which include both read and write access to resources. Furthermore, no work has studied the simulation of the more complex version of the Bell-LaPadula model that includes the current security function, or provided support for limited discretionary features of the Bell-LaPadula model in role-based models.

Dynamic separation of duty

Separation of duty requirements are high level organizational policies, which usually require that sensitive combinations of permissions should not be available to certain users. Separation of duty has always been an important consideration in RBAC models. RBAC introduces separation of duty constraints on the authorization of users to roles [2], which is a means of enforcing such high level policies [65].

Two commonly defined separation of duty constraints in RBAC are *static* separation of duty and *dynamic* separation of duty. The former typically constrains the assignment of users to roles, while the latter constrains the activation of roles in the run-time environment. An example of a dynamic separation of duty constraint is that no user is allowed to activate a particular pair of roles r and r' in the same session.

However, the inheritance of permissions through a role hierarchy may well conflict

with the enforcement of dynamic separation of duty constraints. For example, it is impossible to define a dynamic separation of duty constraint on roles r and r' that have a common senior role r'', because r'' inherits the permissions of both roles. There is no means to assign any user to this common senior role r'' because activation of r'' will violate the dynamic separation of duty constraint with respect to r and r'.

A solution was proposed by Sandhu *et al* [82] which makes a distinction between the permission usage and role activation hierarchies. The resulting model is called ER-BAC96 (Extended RBAC) that has a separate role activation hierarchy which extends the permission usage hierarchy. In this model we can define dynamic separation of duty constraints on roles that have common seniors in the role activation hierarchy, but can not have common seniors in the permission usage hierarchy. However, we believe it is unnecessarily complicated to use two hierarchies to enforce dynamic separation of duty constraints. It is interesting to investigate a simple approach that uses a single role hierarchy and an alternative approach to permission inheritance to address the incompatibility between the role hierarchy and dynamic separation of duty constraints.

1.1.2 Extended RBAC

The role hierarchy is central to many theoretical RBAC models, and serves two different purposes: permission usage and role activation. And as we have seen in the previous section, it is certainly useful to develop an extended RBAC model that distinguishes permission usage and role activation hierarchies. Firstly, it addresses the perceived deficiencies of inheritance with a single role hierarchy, for example, the enforcement of dynamic separation of duty constraints and the simulation of mandatory access control using role-based approach as described above. Secondly, the separation of role activation and permission usage hierarchies allows selective inheritance through hierarchies, which provides a greater flexibility than standard RBAC model in terms

of articulating policies.

The two most significant existing approaches that distinguish between activation and usage hierarchies are the ERBAC96 model [82] and the GTRBAC model [58]. Sandhu introduced the ERBAC96 model that has a separate role activation hierarchy, a relation which is a superset of the permission usage hierarchy. The motivation for making such a distinction is supplied by a consideration of implementing dynamic separation of duty constraints and simulating lattice-based access control [82].

Joshi *et al* developed a generalized temporal (GTRBAC) model [56, 58] that includes three different types of role hierarchies: a role activation hierarchy, a permission usage hierarchy and a general hierarchy that is a combination of the activation and usage hierarchies. The introduction of these three role hierarchies was influenced by the ERBAC96 model and the organizational control principles identified by Moffett and Lupu [67].

We believe that the way in which both ERBAC96 and GTRBAC treat multiple hierarchies suffers from some deficiencies. One of the problems we consider in this thesis is the most appropriate design for multiple hierarchies and, more importantly, the constraints that must be imposed on the relationship between those hierarchies and the authorization semantics induced by those hierarchies.

1.1.3 Spatio-temporal RBAC

In ubiquitous and mobile computing, access control requirements may incorporate some considerations of contextual information, such as the location of the user and the time at which access requests are made. For example, it is natural to limit the hours during which the role of `night-nurse` can be activated by a user, and location in which a user assigned to the role `night-nurse` can access health records.

In the last few years, a number of context-aware RBAC models [13, 14, 28, 43, 49,

58, 76, 86] have been developed to capture these contextual access control requirements. Essentially, all of these models are concerned with the specification of general contextual constraints [28, 43, 86], temporal constraints [13, 58], spatial constraints [14, 49], or spatio-temporal constraints [76] on various components of the standard RBAC model. However, the syntax for these models is rather complicated and the semantics defining the interaction between spatio-temporal constraints and the role hierarchy are not clearly defined. In other words, no existing work has clear authorization semantics in terms of how access requests will be answered in the presence of a role hierarchy and spatio-temporal constraints. Moreover, no existing work assesses the difficulties of implementing these models in practical applications.

We believe that it is important to consider the effect of spatio-temporal constraints on RBAC. In particular, we believe that any spatio-temporal RBAC model should have clear and unambiguous semantics. Hence, it is important to examine the question of how can we develop new spatio-temporal RBAC models with simple syntax and appropriate semantics which explicitly consider the relationship between spatio-temporal constraints and role hierarchies. Having developed spatio-temporal RBAC models, it is also crucial to consider the implementation of these models in practical settings.

1.1.4 Computational problems in RBAC

Just as the development of new RBAC models has led to interesting questions about authorization semantics, new applications and models for RBAC have given rise to a number of interesting computational problems. Joshi *et al* recently raised an important problem – the *inter-domain role mapping* (IDRM) problem [36] – when GTRBAC is employed in distributed environments. Their statement of the IDRM problem is to find a set of roles of minimal cardinality such that the authorized permissions for that set of roles is precisely the set of requested permissions. This problem was further extended

to the *user authorization query* (UAQ) problem [94, 96].

On the other hand, Li *et al* recently studied a number of interesting problems with regard to separation of duty constraints and their enforcement in the context of RBAC [65]. One of the problems – the *role-based static separation of duty constraint* (RSSoD) generation problem – is of particular importance when translating restrictions on permissions expressed in separation of duty constraints to restrictions on role memberships in RBAC. The RSSoD generation problem asks given a set of permissions, find all possible sets of roles such that each set of roles is collectively authorized for the given set of permissions, and any proper subset of that set of roles is not.

However, existing work does not always pose the most appropriate problem, as is the case with the IDRM problem of Du and Joshi [36]. It is easy to show that the IDRM problem is not well-defined, in the sense that many instances of the problem may not have a solution. Moreover, none of the above problems has been properly analyzed and solved. In particular, the IDRM problem and the UAQ problem were conjectured to be **NP**-hard without establishing a formal proof, and directly suggested heuristic algorithms to produce a solution [36, 94, 96]. Nevertheless, if these problems have not been proved to be **NP**-hard, there is no way of ensuring there do not exist efficient algorithms to solve them. In addition, the computational complexity of the RSSoD generation problem has not been established. In summary, we are unaware of any attempt to formally assess the computational complexity of the above problems, which we believe is an important step in understanding and solving the problems.

1.2 Contributions

There is a substantial body of work on RBAC, much of it involving role hierarchies. As we described in the previous section, the consequences of including those hierarchies are

rarely properly explored. In this thesis, we investigate a number of related problems: the perceived limitations of the inheritance semantics of a role hierarchy, the relationship between permission usage and role activation hierarchies, the interaction between spatio-temporal constraints and inheritance, and the computational complexity of several problems arising in hierarchical RBAC. Unlike existing work in these areas, we strive for compatibility with RBAC96 and precise authorization semantics. We now describe the contributions of the thesis, and briefly explain their significance compared with the existing work.

The oriented permission RBAC model (OP-RBAC) developed by Crampton [29] proposed a new mechanism for permission inheritance within a role hierarchy, and was considered to provide more flexibility than standard role-based models. We explore how OP-RBAC can address the perceived shortcomings of the standard RBAC approach to inheritance. In particular, we illustrate that OP-RBAC provides a natural way to simulate a number of Bell-LaPadula models, and to enforce dynamic separation of duty constraints. The simplicity of our approach compares favourably with previous attempts, which typically require multiple role hierarchies and support only limited features of the Bell-LaPadula model.

We then turn our attention to supporting multiple hierarchies in RBAC. We propose a novel extended RBAC model, called ERBAC07, which defines an alternative way to distinguish permission usage and role activation hierarchies. Unlike ERBAC96 and GTRBAC, our approach to defining these hierarchies is based on their inheritance semantics. The syntax we have chosen for ERBAC07 is consistent with RBAC96, and the semantics for the model is extended from our graph-based formalism of RBAC96. In other words, ERBAC07 is compatible with RBAC96 and has a clear authorization semantics.

Having established an extended RBAC model supporting permission usage and role

activation hierarchies, it is natural to then address the question of constraints that may interact with those hierarchies. In particular, we extend the RBAC96 and ERBAC07 models by defining two spatio-temporal constraint specification functions. We use these two functions to define three spatio-temporal RBAC models with different authorization semantics. Compared with existing work, such as GTRBAC and the spatio-temporal model of Ray and Toahchoodee – the only models dealing with multiple role hierarchies and spatio-temporal constraints – our models are concise and have precise authorization semantics. We also propose strategies for the use of our spatio-temporal RBAC models in practical applications, which no existing work has attempted to do.

Finally, we introduce a mathematical framework where we define a number of computational problems that are variations on the standard set cover problem. In particular, we define the *minimal cover problem*, and prove that this problem is **NP**-hard. This result enables us to establish computational complexity for some important problems in the context of RBAC. We also design a number of heuristic algorithms for the minimal cover problem and conduct experiments to evaluate the quality of these algorithms. Our experiment results enable us to identify a good heuristic algorithm with high success rates to compute an exact solution to the minimal cover problem.

We summarize the contributions of this thesis in Table 1.1 that indicates the section in which each contribution appears.

1.3 Structure of the thesis

The remainder of the thesis is organized as follows.

In Chapter 2, we introduce some prerequisite concepts in access control, RBAC and complexity theory. In Section 2.1, we make explicit distinctions between access control policies, states, models and mechanisms, and clarify these distinctions in discretionary

Section 3.3	Simulating the Bell-LaPadula models using OP-RBAC
Section 3.4.1	Supporting dynamic separation of duty in OP-RBAC
Section 3.4.2	Transforming ERBAC96 into OP-RBAC
Section 4.1	A graph-based formalism of $RBAC_1$
Section 4.2	A new model for extended RBAC
Section 4.3	Spatio-temporal RBAC models and trust entities
Section 4.4	Spatio-temporal extended RBAC models
Section 4.5	Practical considerations of spatio-temporal RBAC models
Section 4.6	Spatio-temporal domains
Section 5.2	A new framework for variations on the set cover problem
Section 5.2.3	Complexity of the minimal cover problem
Section 5.2.4	Complexity of the irreducible cover problem
Section 5.3.1	Complexity of the inter-domain role mapping problem
Section 5.3.2	Complexity of the user authorization query problem
Section 5.3.3	Complexity of enforcing separation of duty constraints
Chapter 6	Heuristic algorithms for the minimal cover problem

Table 1.1: Contributions of the thesis

access control and mandatory access control. In Section 2.2, we introduce RBAC, in particular, the RBAC96 family models and the ANSI RBAC standard. We conclude the chapter with a brief outline of complexity theory, which provides a basis for studying computational problems in Chapter 5.

In Chapter 3, we investigate three useful applications of the OP-RBAC model. More specifically, we demonstrate how OP-RBAC can be configured to support a number of different Bell-LaPadule models, dynamic separation of duty constraints, and features of ERBAC96 respectively. In the course of this chapter we examine some existing work: ERBAC96 and GTRBAC, which leads naturally to the material in Chapter 4 where we develop new RBAC models for the distinction between activation and usage hierarchies, and the specification of spatio-temporal constraints.

In Chapter 4, we develop a number of advanced RBAC models that deal with role hierarchies and spatio-temporal constraints. We firstly introduce a graph-based formalism of RBAC96 that defines a simple way of evaluating requests in RBAC96. This graph-based formalism provides a basis for the development of ERBAC07 and

spatio-temporal RBAC models in this chapter. We also suggest some approaches to facilitate the implementation of our spatio-temporal RBAC models.

Joshi *et al* [36] introduced the inter-domain role mapping problem in the context of multiple role hierarchies and temporal constraints. The analysis of this problem motivates us to study some computational problems in Chapter 5. We firstly define some variations on the standard set cover problem, and establish their computational complexity. Then we apply these complexity results to some important problems in RBAC, including the inter-domain role mapping problem, the user-authorization query problem and the separation of duty constraints enforcement problem.

As the minimal cover problem defined in Chapter 5 has been proved to be **NP**-hard, Chapter 6 is concerned with the design and evaluation of heuristic algorithms for solving the problem.

Finally, in Chapter 7, we review the contributions of the thesis and discuss opportunities for future work.

1.4 Publications

A list of publications describing some of the research results contained in this thesis is provided below.

- L. Chen and J. Crampton. Applications of the oriented permission role-based access control model. In *proceedings of the 26th IEEE International Performance Computing and Communications Conference*, pages 387-394, 2007.

- L. Chen and J. Crampton. Inter-domain role mapping and least privilege. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, pages 157-162, 2007.

- L. Chen and J. Crampton. On spatio-temporal constraints and inheritance in role-based access control. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*, pages 205-216, 2008.

- L. Chen and J. Crampton. Set covering problems in role-based access control. In *Proceeding of the 14th European Symposium on Research in Computer Security*, pages 689-704, 2009.

Chapter 2

Background

The main purpose of this chapter is to introduce relevant background material. In Section 2.1 we give a brief introduction to some basic concepts and earlier developments in access control. In particular, we introduce the two best known types of access control – discretionary access control and mandatory access control – which serve as a motivation for role-based access control and inform the material in Chapter 3. In Section 2.2 we introduce role-based access control and review the RBAC96 family of models and the ANSI RBAC standard, which is fundamental to the remainder of this thesis. We conclude the chapter with a brief outline of complexity theory that will be required in Chapters 5 and 6 when we study some computational problems in the context of role-based access control.

2.1 Access control

One of the essential security services in (multi-user) computer systems is *access control*, a mechanism for constraining the interaction between (authenticated) users and protected resources. In the context of computer systems, access control may also be referred to as *authorization*. Generally, access control is concerned with controlling

which users have access to which resources in computer systems. Given a computer system, access control can be implemented in many places and at different levels [3, 39]. Operating systems implement access control to limit access to files, directories and devices. Database management systems apply access control to regulate access to tables and views. Sensitive applications incorporate access control to ensure some application functions are only available to certain users and other applications. Moreover, access control can take many forms, which means in addition to checking whether a user is authorized to access a resource, access control may also be concerned with constraining when, where and how a resource can be used. For example, in the context of the financial sector, (i) financial managers might have access to some sensitive reports only during working hours and at certain offices, and (ii) to prevent fraud, we may require that a financial manager is not allowed to approve a loan that was created by herself.

When implementing access control in computer systems, it is important to understand and distinguish four concepts: *access control policies*, *access control states*, *access control models* and *access control mechanisms* [32, 34, 78]. Access control is often "policy-based" in the sense that a user's request to access a resource (an *access request*) is checked to see if it is authorized by a policy. Access control policies generally specify who is authorized to access which resources under what circumstances. In other words, access control policies define rules for deciding whether access requests should be granted or not. The conditions that determine whether an access request is authorized are usually defined in terms of the (access control) state or configuration of the computer system. The state of the system is a snapshot of all security-relevant information at a point in time; the state may change over time. Consider, for example, the simple security property defined in the Bell-LaPadula access control model [12], which says that a user u is authorized to read a document d only if $\lambda(u) \geqslant \lambda(d)$, where $\lambda : U \cup D \to L$ is a labeling function and $(L, \leqslant)$ is a lattice of security labels. Infor-

mally, the policy states that a user is authorized to read a document only if the user's security level is at least as high as that of the requested document. The state is defined by the security lattice L and the security function λ. Crampton recently defined the distinction between access control policy and state as shown in the following quote [32].

> *"An access control policy is a specification of decision-marking function that takes a request query and access control state as inputs and returns an access control decision."*

By definition, the evaluation of an access control policy is dependent on the access request and the current access control state of the system. An obvious example is the Chinese Wall policy [18], which is designed to prevent a user from accessing documents whose owners belonging to a conflict interest class. The evaluation of this policy depends on the mutable system state: namely, historical information about which documents the user have previously accessed.

In summary, it is important to make a clear distinction between policy and state, because such distinction provides reuse of already existing authorization decision functions (policies) and separates the specification of access control state from policy semantics. We use the term *authorization syntax* for the language that is used to specify states, and *authorization semantics* for the effect of evaluating a policy for a given request and state. Equivalently, we could regard the semantics of a state (with respect to a fixed policy) to be the set of requests that are authorized by the state.

An access control model is an abstract mathematical description of authorization syntax, together with a definition of authorization semantics. More specifically, it defines a collection of sets, functions and relations that provide a method for encoding access control states, and specifies the conditions that must be satisfied for an access request to be granted [31]. The RBAC96 model is a typical example of an access

control model, which we will introduce in Section 2.2. Informally, an access control model provides a blueprint for the implementation of access control systems. It may also provide assurances of security for an implemented system, an example being the famous security theorem of the Bell-LaPadula model [12].

An access control mechanism (also known as a *reference monitor* or an *authorization service*) is a computer function that implements the controls formally stated in the access control model. However, it is often difficult to provide an access control mechanism that correctly and completely implements the access control model. In general, any access control mechanism includes two distinct components: the *authorization enforcement function* (AEF) and the *authorization decision function* (ADF) [51].[1] Figure 2.1 illustrates the generic architecture for an access control mechanism. The AEF intercepts all access requests and forwards them to the ADF. The ADF decides whether an access request is authorized by consulting relevant access control states, and returns a binary decision (grant or deny) to the AEF. The AEF then enforces that decision by either making the resource available to the requestor or letting the requestor know the access is unauthorized. In short, the AEF is responsible for ensuring that every access request is evaluated to determine whether it is authorized, while the ADF is an implementation of a access control policy [32], which queries the access control states and decides whether the request is authorized.

The Trusted Computer System Evaluation Criteria (TCSEC), commonly known as the "Orange Book" [89], defines two fundamental types of access control: *discretionary access control* and *mandatory access control*, which are widely used in commercial and government sectors today. We now briefly introduce discretionary and mandatory access controls, and present models and mechanisms associated with them.

[1]The eXtensible Access Control Markup Language (XACML) 2.0 specification [71] introduced the terms *policy enforcement point* (PEP) and *policy decision point* (PDP) which are equivalent to AEF and ADF respectively.

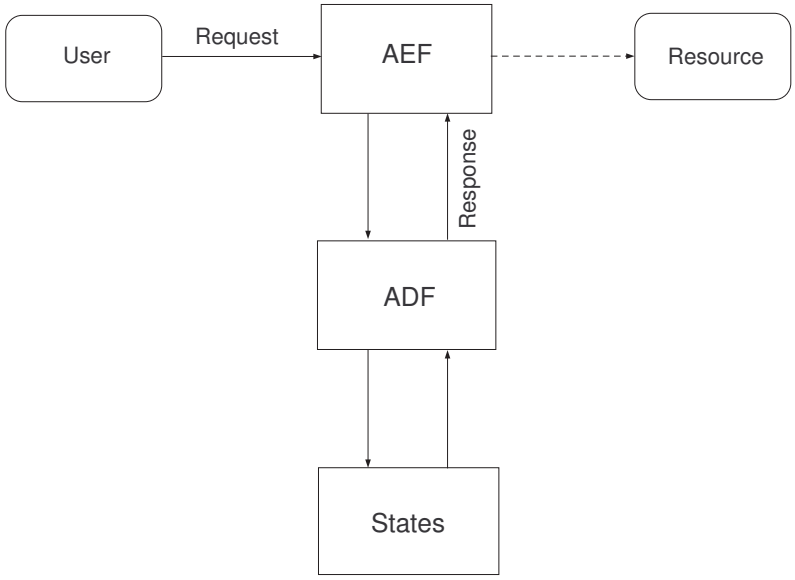


Figure 2.1: The generic architecture of an access control mechanism

2.1.1 Discretionary access control

Discretionary access control usually requires each resource to have an *owner*, and the owner of a resource is able to authorize access to the resource for other users [89]. In other words, an individual user is able to decree who is allowed to have access to the resources she owns. A request to access resources is evaluated based on the identity of the requesting user or the group to which the user belongs. Therefore, discretionary access control is also called *identity-based access control* [17].

The first discretionary access control model was introduced by Lampson [63], and was further refined by Graham and Denning [48]. The Harrison-Ruzzo-Ullman (HRU) model [50] is the most widely known discretionary access control model and provides a basis for subsequent research [79]. All these discretionary access control models introduce the formal notions of *subject*, *object* and *access right*, and use an *access control matrix* as a structure to represent access control states.

A subject is usually an active system entity that initiates requests to perform some actions on resources. The subjects are generally considered to be users, but more

precisely, they are processes or threads that execute under the control of a computer system. An object is usually a passive system entity that can be any resource to which access should be retrieved. Typical examples of objects in operating systems include files, directories and printers. An access right is an action that is invoked by subjects on objects. Typical examples of access rights in an operating system include read, write and execute.

An access control matrix M (also known as a *protection matrix*) has rows indexed by subjects and columns indexed by objects. The matrix entry for subject s and object o, denoted $M_{s,o}$, contains a set of access rights for which s is authorized with respect to o. An access request is modeled as a triple (s, o, a), and is authorized if, and only if, $a \in M_{s,o}$. Table 2.1 represents a simple access control matrix, where the file `diary.doc` can be read and written by `Bob` while `Alice` has no access at all. In addition, `Bob` is an owner of `diary.doc` and he can allow Alice to read `diary.doc` by entering the right `read` into the entry [`Alice`, `diary.doc`] in the matrix.

	bill.doc	diary.doc	install.exe
Bob	{`read`}	{`own`, `read`, `write`}	{`execute`}
Alice	{`read`, `write`}	–	{`execute`}

Table 2.1: An access control matrix

An access control matrix will become very large and the non-empty entries will be sparse in a computer system with large numbers of users and resources. In this case, it might not only cause performance problems but also be vulnerable to administrative errors. Therefore, an access control matrix is rarely implemented in a computer system. Instead, two popular discretionary access control mechanisms store the access control matrix either by columns (access control lists) or rows (capability lists).

An *access control list* is associated with an object, and consists of zero or more access control entries. Each *access control entry* specifies a subject and the set of access rights

for which that subject is authorized. In contrast, a *capability list* is associated with a subject, and contains a list of permissions. Each permission identifies an object and a set of access rights that have been assigned to the subject for that object. In Figure 2.1, for example, principal `Bob` has the capability to `read` object `bill.doc`. In other words, an access control list is concerned with what may be done with an object. In modern operating systems, access control lists are commonly used to protect files and resources. On the other hand, a capability list is concerned with what a subject is allowed to do. It is usually incorporated in application-oriented IT systems that focus on controlling the actions of subjects.

2.1.2 Mandatory access control

Mandatory access control is deployed when the use of resources is determined by the characteristics of the resource and the subject, not the wishes of the owner. The characteristics of the subject and the object are often represented by *security levels* assigned to subjects and objects in the system. The security level of a subject, also called the *clearance level* of the subject, reflects the level of trust assigned to the subject, while the security level of an object, also called the *classification level* of the object, reflects the level of sensitivity of the contents of the object. The set of security levels is partially ordered and is often assumed to form a lattice [80]. Thus, a computer system that implements mandatory access control is often called a *multi-level secure system*.

The act of accessing an object can be regarded as initiating an information flow. In particular, read access can be seen as a flow of information from object to subject, while write access is a flow of information from subject to object. An mandatory information flow policy for confidentiality requires that high level information cannot flow to a lower level.[2] In other words, the information flow policy requires that a subject is only allowed

[2]To safeguard the integrity of information, Biba proposed a dual policy that low level integrity

to read an object if the subject's security level is at least as high as that of the object, and a subject is only allowed to write an object if the object's security level is at least as high as that of the subject. These two requirements are usually called "no-read-up" policy and "no-write-down" policy, respectively. In practice, in order to define all the authorization requirements of a computer system, mandatory information flow policy is usually augmented with a discretionary access control policy that is defined by the object owners and implemented using a protection matrix [12].

The Bell-LaPadula model has been the subject of extensive research in several seminal papers [8, 10, 11, 12] and is perhaps the best known of all access control models. It provides a combination of mandatory access control and discretionary access control. We formally introduce a simplified version of the Bell-LaPadula model [10, 11] by defining a set of subjects S, a set of objects O, an access control matrix M with columns indexed by O and rows indexed by S, a partially ordered set of security labels $(L, \leqslant)$, a security function $\lambda : S \cup O \rightarrow L$[3], and a set of access rights $A = \{\texttt{read}, \texttt{append}, \texttt{write}\}$, where `read` denotes read only access, `append` denotes write only access and `write` denotes both read and write access. The *state* of a Bell-LaPadula system is defined to be (V, M), where V represents the set of *active triples* of the form (s, o, a), where $s \in S$, $o \in O$, and $a \in \{\texttt{read}, \texttt{append}, \texttt{write}\}$; that is, the set of access requests that have been granted by the access control mechanism [10].

To enforce a mandatory information flow policy and a discretionary access control policy, every state of a Bell-Lapadula system must satisfy three security properties: the *simple security property*, the **-property* and the *discretionary security property*.

- A state (V, M) satisfies the simple security property if for all $(s, o, \texttt{read}) \in V$,

information is not allowed to flow to a higher level [16]. In this chapter, we only consider information flow policies in the Bell-LaPadula model, because the formulation of the Biba integrity model is the dual of that in the Bell-LaPadula model.

[3]We assume that the labelling of entities is performed by a single security function λ, and the security function λ is fixed

$\lambda(s) \geqslant \lambda(o)$. In other words, the simple security property is satisfied if every granted read access (that is, belongs to V) was authorized by the "no-read-up" policy.

- A state (V, M) satisfies the *-property if for all $(s, o, \texttt{append}) \in V$, $\lambda(s) \leqslant \lambda(o)$. In other words, the *-property is satisfied if every granted append access was authorized by the "no-write-down" policy.[4]

- A state (V, M) satisfies the discretionary security property if for all $(s, o, a) \in V$, $a \in M_{s,o}$. In other words, every access request that has been granted was authorized by an appropriate entry in the access control matrix.

In actual fact, the Bell-LaPadula model [12] makes use of three security functions to label subjects and objects, and the simple security property and the *-property are defined in terms of those three security functions. We will introduce a more complete version of the Bell-LaPadula model in Chapter 3 when studying the simulation of the Bell-LaPadula models using role-based methods.

The Bell-LaPadula model has been implemented in military applications and commercially sensitive applications [17]. Multics is considered to be the best example of an operating system built for security [17], in which the protection mechanisms basically implement the Bell-LaPadula model.

2.2 Role-based access control

Role-based access control has received considerable attention in recent years, and is widely accepted as an alternative to discretionary and mandatory access controls [39].

[4]The *-property in the original formulation of the Bell-LaPadula model [10] requires that for all $(s, o, \texttt{read}) \in V$, and for all $(s, o', \texttt{append}) \in V$, $\lambda(o) \leqslant \lambda(o')$. The *-property we introduce here is the most commonly accepted version (see [73, 80]), and is slightly stronger than the *-property defined in the original Bell-LaPadula model [10]. In Chapter 3, we consider more complex version of the *-property.

The motivation for the development of role-based access control is to address the perceived deficiencies of existing discretionary and mandatory access control models in terms of specification and enforcement of organization-specific access control policies, and to reduce the complexity and cost of administering systems based on these models [44]. In other words, neither discretionary nor mandatory access control is sufficiently suitable for the needs of most commercial systems. More specifically, discretionary access control, for example, permits users to grant or revoke access to any of the objects they owned. However, for many organizations within industry and civilian government, the corporation or agency is the owner of system objects, rather than the end users [44]. Hence, it is not appropriate to allow users to pass access rights on the objects to other users in these organizations. Mandatory access control that focuses on preserving confidentiality is too restrictive and therefore inappropriate for these organizations as well. In addition, in an organization with a large and dynamic user population, it is time-consuming and error-prone to manage an access control system based on access control lists. In particular, it is extremely difficult to revoke a user's access permissions, because it involves checking the access control lists of all objects in the system.

The central concept of role-based access control is that of a *role* that can be seen as a job or position within an organization. Role-based access control also introduces the notion of a *permission* or *privilege* that refers to some combination of access rights and objects. The internal structure of a permission depends on the implementation details of a role-based access control system [81]. Informally, roles form an intermediate layer between users and permissions. More specifically, permissions are associated with roles based on work-related activities, and users are assigned to roles based on their job duties, qualifications or competencies. Figure 2.2 shows the relationship between users, roles and permissions, where double-headed arrows indicate a many-to-many

relationship. For example, a user can be assigned to one or more roles, and a role can have one or more user members. This arrangement of controlling access through roles provides great flexibility and simplifies the management of access controls. For example, within a large organization, as job assignments and organizational functions change, we can simply adjust user-role association and permission-role association respectively, rather than allocating permissions to each user on an individual basis.

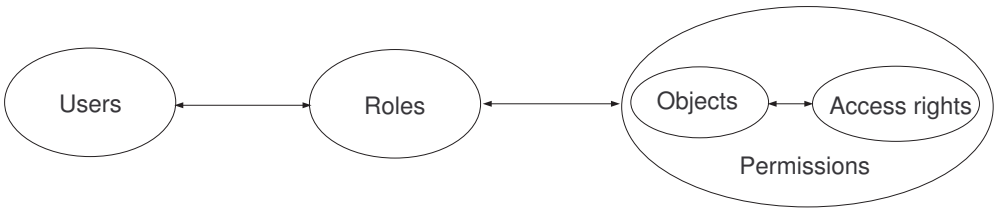


Figure 2.2: A simple relationship between users, roles and permissions

Role-based access control has been the subject of considerable research in the last decade, resulting in the development of a number of different role-based access control models. Ferraiolo and Kuhn developed the first formal role-based access control model in 1992 [40], and then introduced further refinement in 1995 [38]. Nyanchama and Osborn proposed a role graph model [70] that organizes roles using a graph, and the role graph model has led to subsequent research over the years [92, 93]. The RBAC96 family of models [83], due to Sandhu *et al*, is undoubtedly the most well known model for RBAC, and provides the basis for the recent ANSI RBAC standard [2]. In this thesis, we will focus on the RBAC96 model and the ANSI RBAC standard.

2.2.1 RBAC96

The RBAC96 family of models consists of four conceptual models that form a hierarchy as shown in Figure 2.3. The simplest model, $RBAC_0$, introduces the basic features of role-based access control. $RBAC_1$ and $RBAC_2$ extend $RBAC_0$ through the addition of role hierarchy and constraints respectively. $RBAC_3$ includes all the features of $RBAC_1$

and RBAC_2.

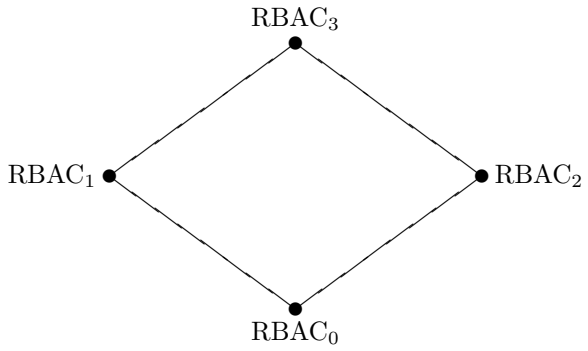


Figure 2.3: The RBAC96 family of models

RBAC_0

RBAC_0 defines a set of users U, a set of roles R, a set of permissions P, a user-role assignment relation $UA \subseteq U \times R$ and a permission-role assignment relation $PA \subseteq P \times R$. We refer to such sets and relations as *components* of RBAC_0. We write $\mathsf{Roles}(u)$ for the set of roles to which a user u is explicitly assigned by the UA relation; that is, $\mathsf{Roles}(u) = \{r \in R : (u, r) \in UA\}$. Similarly, we write $\mathsf{Roles}(p)$ for the set of roles to which a permission p is explicitly assigned by the PA relation; that is, $\mathsf{Roles}(p) = \{r \in R : (p, r) \in PA\}$. Given $r \in R$, we write $\mathsf{Prms}(r)$ to denote the set of permissions for which r is explicitly assigned, and for $R' \subseteq R$, we write $\mathsf{Prms}(R')$ to denote the set of permissions for which the roles in R' are explicitly assigned. That is,

$$\mathsf{Prms}(r) = \{p \in P : (p, r) \in PA\} \quad \text{and} \quad \mathsf{Prms}(R') = \bigcup_{r \in R'} \mathsf{Prms}(r).$$

RBAC_0 also introduces the concept of a *session* that is synonymous with a subject in traditional access control models. When a user logs in to a computer system (employing a role-based access control mechanism), the user can establish a session during which she can activate a subset of roles to which she is explicitly assigned. A user may run multiple sessions simultaneously, and each session may have a different combination

of active roles. We denote the set of sessions by S, and the user who established the session $s \in S$ by $\mathsf{User}(s)$. We write $\mathsf{Roles}(s)$ for the set of roles activated in a session s, that is, $\mathsf{Roles}(s) \subseteq \mathsf{Roles}(u)$, where $u = \mathsf{User}(s)$. A request by u to invoke permission p in session s is granted if u has activated one of p's explicitly assigned roles in s, that is, $\mathsf{Roles}(s) \cap \mathsf{Roles}(p) \neq \emptyset$.

$\mathbf{RBAC}_1$

RBAC_1 introduces the concept of a *role hierarchy*, which is modeled as a partial order on the set of roles. In other words, the role hierarchy is a binary relation $RH \subseteq R \times R$ that is reflexive, anti-symmetric and transitive. We write $r \leqslant r'$ if $(r, r') \in RH$ and $r < r'$ if $r \leqslant r'$ and $r \neq r'$. The role hierarchy is used to reduce the administrative burden by reducing the number of explicit assignments in the UA and PA relations. That is, if $(u, r) \in UA$ and $r' \leqslant r$, then u is implicitly assigned to r'; and if $(p, r) \in PA$ and $r \leqslant r'$ then p is implicitly assigned to r'.

Given $r \in R$, we write $\downarrow r$ to denote the set of all elements in R that are less than or equal to r: that is, $\downarrow r = \{r' \in R : r' \leqslant r\}$. Similarly, we write $\uparrow r = \{r' \in R : r \leqslant r'\}$. We write $\downarrow\mathsf{Roles}(u)$ to denote the set of roles explicitly and implicitly assigned to u, that is

$$\downarrow\mathsf{Roles}(u) = \{r' \in R : \exists r \in \mathsf{Roles}(u), r' \leqslant r\}.$$

Similarly, we write $\uparrow\mathsf{Roles}(p)$ to denote a set of roles explicitly and implicitly assigned to p, that is

$$\uparrow\mathsf{Roles}(p) = \{r' \in R : \exists r \in \mathsf{Roles}(p), r \leqslant r'\}.$$

Note that henceforth we will write "authorized" to mean "explicitly and implicitly assigned". We write $\mathsf{Prms}(\downarrow r)$ for the set of permissions for which r is authorized.

We now analyze the use of sessions in RBAC_1. A user u is allowed to establish

a session s to activate a set of roles that can be any subset of roles for which she is authorized, that is, $\mathsf{Roles}(s) \subseteq \downarrow\mathsf{Roles}(u)$, where $u = \mathsf{User}(s)$. A request by u to invoke permission p in session s is granted if u has activated one of p's authorized roles in s, that is $\mathsf{Roles}(s) \cap \uparrow\mathsf{Roles}(p) \neq \emptyset$. Providing that $RH = \{\}$, we have $\mathsf{Roles}(s) \subseteq \downarrow\mathsf{Roles}(u) = \mathsf{Roles}(u)$ and $\uparrow\mathsf{Roles}(p) = \mathsf{Roles}(p)$. In this case, we recover RBAC_0 authorization semantics. In other words, RBAC_0 is a special case of RBAC_1 in which the hierarchy relation is empty.

$\mathbf{RBAC}_2$

RBAC_2 extends RBAC_0 by introducing constraints, which usually specifies a set of forbidden configurations of an access control system. RBAC_2 informally discussed three types of constraints that can be defined over the components of RBAC_0: *dependency constraints*, *cardinality constraints* and *separation of duty constraints*. Dependency constraints may specify that certain roles can be activated only if the requesting user has already activated some other roles. Cardinality constraints may specify the maximum number of users that can be assigned to or activate a role, and the maximum number of roles that can be activated in a session by a user. Dependency and cardinality constraints in role-based access control have received little attention in the literature, and support for these constraints was dropped in the ANSI RBAC standard.

Separation of duty is a widely recognized business principle that is used to prevent conflict of interests arising or to prevent fraudulent actions. It requires that more than one user be involved in the execution of two or more tasks in a business process, which, if performed by the same user, could expose the process to misuse. Early work on separation of duty constraints in computer systems includes the Chinese Wall security policy, which prohibits any user from having access to documents belonging to two different competitors [18]. In RBAC_2, separation of duty constraints are supported by

defining *mutually exclusive roles*. Generally, the set of permissions that are required to perform sensitive tasks are assigned to mutually exclusive roles, and no user can be assigned to more than one role in the mutually exclusive set, or no user is allowed to activate more than one role in the set. Simon and Zurko [84] refer to the former as *static separation of duty*, and to the latter as *dynamic separation of duty*. The research community has since taken an active interesting in proposing specification schemes for separation of duty constraints in role-based access control [1, 15, 30, 45, 52, 70, 84], and suggesting models for enforcing such constraints [15, 30, 65, 70, 84].

$\mathbf{RBAC_3}$

RBAC_3 incorporates the features of RBAC_1 and RBAC_2. It is known that there exists a certain "tension" between separation of duty constraints and a role hierarchy. In the simplest case, if p and q are mutually exclusive permissions, then p and q should be assigned to two different incomparable roles r and r'. The separation of duty between two roles r and r' is impossible to realize if r and r' have a common senior role r'', because r'' inherits the permissions of both roles. If we require that no user is allowed to be assigned to or activate r'', then there is no point in having the "unassignable" role r'' in the role hierarchy. A number of extended role-based access control models, which we discuss in Chapter 3 and 4, have been introduced to address this shortcoming of RBAC_3.

2.2.2 ANSI-RBAC

The American National Standard Institute (ANSI) standard for role-based access control (RBAC) [2] provides a consistent and uniform definition of RBAC features, and gives information technology vendors a guideline for designing RBAC products. The ANSI RBAC standard includes three components: Core RBAC, Hierarchical RBAC,

and Constrained RBAC.

Core RBAC defines the basic building blocks of a RBAC system: a set of basic element sets U, S, R and P, a set of relations UA and PA, and a set of mapping functions shown in the top part of Table 2.2. The table uses ANSI RBAC syntax rather then the RBAC96-style syntax we used in the previous section, but we can easily extend our notation to define these functions if desired.

Core Component
$assigned\_users(r) = \{u \in U : (u, r) \in UA\}$
$assigned\_permissions(r) = \{p \in P : (p, r) \in PA\}$
$session\_users(s) = u$
$session\_roles(s) \subseteq \{r \in R : (session\_users(s), r) \in UA\}$
$avail\_session\_perms(s) = \bigcup_{r \in session\_roles(s)} assigned\_permissions(r)$
Hierarchical RBAC
$authorized\_users(r) = \{u \in U : r \leqslant r', (u, r') \in UA\}$
$authorized\_permissions(r) = \{p \in P : r \geqslant r', (p, r') \in PA\}$
Proposed extensions for Hierarchical RBAC
$session\_roles(s) \subseteq \{r \in R : r \leqslant r', (session\_users(s), r') \in UA\}$
$avail\_session\_perms(s) = \bigcup_{r \in session\_roles(s)} authorized\_permissions(r)$

Table 2.2: ANSI-RBAC mapping functions

Hierarchical RBAC introduces role hierarchies that are intended to reflect the hierarchical nature of many organizations and thereby simplify access control management. There are two types of role hierarchies defined in the hierarchical component: *general role hierarchies* and *limited role hierarchies*. The general role hierarchy is an arbitrary partial order on the set of roles R. The limited role hierarchy is defined to be an inverted tree structure, where each role has only a single immediate child. It is interesting to note that separation of duty constraints are compatible with the limited role hierarchy: the separation of duty constraint between two incomparable roles r and r' can be enforced, because there does not exist a common senior role r'' of r and r' in the limited role hierarchy.

In addition, the hierarchical component defines two functions *authorized_users* and *authorized_permissions*, shown in the second section of Table 2.2. The standard states that $r \leqslant r'$ only if $authorized\_permissions(r) \subseteq authorized\_permissions(r')$ and $authorized\_users(r') \subseteq authorized\_users(r)$. The core component defines functions *avail_session_perms* and *session_roles*, shown in the first section of Table 2.2, but no analogous function is defined for the hierarchical component, which is a curious omission. We propose new definitions for the functions *session_roles* and *avail_session_perms* for the hierarchical component. These definitions are shown in the third section of Table 2.2. Note that in core RBAC, $assigned\_permissions(r) = authorized\_permissions(r)$ for all r. Hence, our definition is consistent with that given in the core component.

Constrained RBAC introduces two types of separation of duty constraints: Static Separation of Duty (SSD) and Dynamic Separation of Duty (DSD). A SSD constraint is specified as a pair (R', n), where $R' \subseteq R$ and $2 \leqslant n \leqslant |R'|$, and can be defined in Core RBAC and Hierarchical RBAC. The SSD constraint (R', n) is satisfied by Core RBAC if no user is *assigned* to n or more roles in the set R', while the SSD constraints (R', n) is satisfied by hierarchical RBAC if no user is *authorized* for n or more roles in the set R'. Like SSD, a DSD constraint is written as (R', n), where $R' \subseteq R$ and $2 \leqslant n \leqslant |R'|$, which limits the roles that a user can activate in one session. Specifically, the DSD constraint is satisfied if no user may simultaneously activate n or more roles from R' in one session. Clearly, satisfaction of a SSD constraint is simply enforced by checking the number of roles in R' for which each user is authorized. Similarly, it is simple to check whether a DSD constraint is satisfied by computing the number of roles in R' that is activated in each session.

Li *et al* recently identified a number of design flaws and technical errors for the ANSI RBAC standard and suggested some improvements to the standard [64]: the standard

should remove the notion of sessions from the core component, and specify only one role can be activated in a session; the standard should introduce a new approach for modelling the role hierarchy so as to facilitate the changes to the role hierarchy; and the standard should clearly specify and discuss the semantics of role hierarchy in terms of user inheritance, permission inheritance and activation inheritance. The authors of the original proposal for the ANSI-RBAC standard responded to each suggestion and clarified the rationale for the choices they made in the standard [41].

2.3 Complexity theory

In this section, we give a brief introduction to compleixity theory, which provides a basis for studying some fundamental problems in role-based access control in Chapter 5 and Chapter 6.

The study of computational problems is one of the main subjects in theoretical computer science. In general, a (computational) *problem* ϕ can be expressed in terms of some relation $\phi \subseteq I_\phi \times S_\phi$, where I_ϕ is the set of *problem instances* and S_ϕ is the set of *problem solutions*. A (deterministic) *algorithm* is said to *solve* a problem ϕ if that algorithm is guaranteed always to produce a solution for any instance i of ϕ. Given a problem ϕ, clearly, we are interested in finding the most efficient algorithm for solving the problem. The efficiency of an algorithm can be measured by the time and memory required to execute the algorithm. In this thesis, we only concentrate on time requirements when measuring the efficiency of algorithms.

The *worst-case time complexity* for an algorithm is a function $\psi : \mathbb{N} \to \mathbb{R}$ that expresses the largest number of operations needed by the algorithm to solve a problem instance of size n.[5] The *average-case time complexity* of an algorithm is described in

[5]We assume the time complexity of an algorithm is independent of the encoding of the input and the underlying computation model that executes the algorithm.

terms of the average number of operations needed by the algorithm to solve all problem instances of size n. In this thesis, we shall concentrate on finding only worst-case time complexity for algorithms. The (worst-case) time complexity of an algorithm is often a complex expression, which is simplified by using "big-O" notation that only considers the highest order term of the expression, and discards both the coefficient of that term and any lower order terms. We formalize this notation in the following definition (see [42], for example).

Definition 2.3.1 *Let f and g be functions $f, g : \mathbb{N} \to \mathbb{R}^+$. We say $f(n) = \mathcal{O}(g(n))$ if there exist positive integers c and n_0 such that $f(n) \leqslant c \cdot g(n)$ for all $n \geqslant n_0$.*

For instance, an algorithm that uses $50n^3 + 20n^2 + n$ operations to solve a problem instance of size n has time complexity $\mathcal{O}(n^3)$. With the help of big-O notation, we can determine whether it is practical to use a particular algorithm to solve a problem as the size of the problem instance increases.

A *polynomial time algorithm* is defined to be one whose time complexity function is $\mathcal{O}(p(n))$ for some polynomial function p, where n denotes the size of the input to the algorithm. Polynomial time algorithms are normally regarded as desirable algorithms, and henceforth we may refer to polynomial time algorithms as efficient algorithms. A problem is said to be *tractable* if there is a polynomial time algorithm that can solve the problem.

A problem is said to be *intractable* if there is no polynomial time algorithm that can solve the problem. Given a hard problem, proving its intractability is just as hard as finding a polynomial time algorithm for it. However, the theory of **NP**-completeness [27, 42, 60] provides techniques for proving that the given problem is as hard as some problem for which no efficient algorithm is believed to exist, despite extensive research. Indeed, there exists a list of complexity classes, including **P**, **NP**,

NP-complete and **NP**-hard, where each class identifies a set of problems of related time complexity.

Reduction is a basic tool for relating the time complexities of different problems. Basically, a reduction from a problem ϕ to a problem ϕ' presents a method for solving ϕ using an algorithm for ϕ'. Before defining two important types of reductions, we introduce the concept of a decision problem on which the complexity classes **P**, **NP** and **NP**-complete are based.

We define a *decision problem* ϕ to be a predicate $\phi : I_\phi \rightarrow \{0, 1\}$. In other words, every instance of the decision problem has one of two solutions. A decision problem ϕ is said to be in **P** if there exists a polynomial time algorithm that solves ϕ.

In order to define **NP** we introduce the concept of a *nondeterministic algorithm*. A nondeterministic algorithm for a decision problem ϕ takes a problem instance $i \in I_\phi$ as input, and executes the following two stages: (i) nondeterministically guess a structure S (also called *certificate*) from i, and (ii) verify deterministically whether S can prove that the answer for i is 1. The algorithm outputs "yes" if there exists S that proves that the answer for i is 1 and outputs "no" otherwise. The computation of an nondeterministic algorithm is a tree whose branches correspond to different possible guesses, and each independent guess is verified concurrently. The time complexity of a nondeterministic algorithm is defined to be the time used by the longest computation branch [85], that is the largest number of operations used to verify a particular guess. A nondeterministic algorithm is said to solve a decision problem ϕ in polynomial time if there exists a polynomial p such that, for any problem instance $i \in I_\phi$ of size n, there exists at least one guess S which leads the algorithm to return "yes" with the time complexity $\mathcal{O}(p(n))$ if and only if the answer for i is 1.[6] A decision problem ϕ is said

[6]Clearly, this definition imposes that the size of the guessed structure S is polynomially bounded, because the algorithm should be able to check that guess in polynomial time.

to be in **NP** if it can be solved by a polynomial time nondeterministic algorithm.

Consider the *set cover problem*: given a universe U, a collection $\mathcal{C}$ of subsets of U such that $U = \bigcup_{C \in \mathcal{C}} C$, and an integer k, does $\mathcal{C}$ contain a *cover* of U having size k or less, that is, a subset $\mathcal{D} \subseteq \mathcal{C}$ such that $\bigcup_{D \in \mathcal{D}} D = U$ and $|\mathcal{D}| \leqslant k$? There is no known polynomial time algorithm for solving the set cover problem. We can easily obtain an exponential time algorithm for this problem by searching every possible subset of $\mathcal{C}$ until one with the desired property is found. However, we can construct a nondeterministic algorithm that simply guesses a subset $\mathcal{D}$ of $\mathcal{C}$ and checks in polynomial time whether the union of $\mathcal{D}$'s elements equals U and whether $\mathcal{D}$ has no more than k elements. Clearly, for any instance $(U, \mathcal{C}, k)$ of the set cover problem, there will exist a guess that leads the nondeterministic algorithm to produce an output of "yes" if and only if there exists a cover for the instance $(U, \mathcal{C}, k)$. Therefore, the set cover problem is in **NP**.

The question of whether **P** = **NP** is one of the greatest unsolved problems in theoretical computer science. It is easy to see that **P** $\subseteq$ **NP**, because a deterministic algorithm is just a special case of a nondeterministic one, in which no guess is performed [42]. However, it is not known whether **NP** $\subseteq$ **P**. Most researchers believe that **NP** $\nsubseteq$ **P**, because no polynomial time algorithm has been found for certain problems in **NP** such as the set cover decision problem. The concept of ***NP-completeness*** is very useful when considering the question of whether **P** = **NP**. Informally, the **NP**-complete problems are the "hardest" problems in **NP** in the sense that they are the ones most likely not to be in **P**.

Definition 2.3.2 *Given two decision problems ϕ and ϕ', we say there is a* polynomial transformation *from ϕ to ϕ' (written $\phi \propto \phi'$) if there is a polynomial time function $f : I_\phi \to I_{\phi'}$ such that for all $i \in I_\phi$, $\phi(i) = 1$ if and only if $\phi'(f(i)) = 1$.*

A decision problem ϕ is said to be **NP**-complete if ϕ is in **NP**, and for every

problem ϕ' in **NP** there exists a polynomial transformation of ϕ' to ϕ. Since polynomial transformation is transitive, we can also say that a decision problem ϕ is **NP**-complete if ϕ is in **NP**, and there exists a polynomial transformation from a known **NP**-complete problem ϕ' to ϕ. If an **NP**-complete problem can be solved by a polynomial time algorithm, then all the problems in **NP** are tractable. Thus, the question of whether **P** = **NP** is reduced to the question of whether **NP**-complete problems are tractable.

The techniques for proving **NP**-completeness can be used to prove the hardness for some problems outside **NP**. The basic idea is to generalize the notion of the polynomial transformation in such a way that a set of problems other than decision problems can be shown at least as hard as the **NP**-complete problems.

Definition 2.3.3 *An* oracle *for problem ϕ is an abstract device that is capable of returning a solution for any instance of ϕ. It is assumed that the oracle returns the solution in just one computational step.*

Definition 2.3.4 *Given two problems ϕ and ϕ', we say*

- *there is a* polynomial time Turing reduction *from ϕ to ϕ' (written $\phi \propto_T \phi'$) if there is a polynomial time algorithm f which solves ϕ by querying an oracle for ϕ'.*

- *ϕ and ϕ' are* polynomial time Turing equivalent *(written $\phi =_T \phi'$) if $\phi \propto_T \phi'$ and $\phi' \propto_T \phi$.*

We are not concerned with the way the oracle determines its responses. We can imagine the oracle for ϕ' as a subroutine someone gives to us. We don't know how it works, we just know what it returns. What we need to do is to write a program f to solve ϕ in polynomial time by invoking the subroutine for ϕ' many times in the program f (though no more than a polynomially bounded number of times). Now we

say that a (general) problem ϕ is **NP**-hard if there exists a polynomial time Turing reduction from a **NP**-complete problem ϕ' to ϕ. If there is a polynomial time algorithm for any **NP**-hard problem, then there are polynomial time algorithms for all problems in **NP**, and hence **P** = **NP**. Note that, if **P** = **NP**, it does not mean that all **NP**-hard problems are tractable, because some **NP**-hard problems may be harder than **NP**-complete problems.

However, it is important to prove that the hard problem is **NP**-complete or **NP**-hard, because it provides a better understanding of the problem and leads algorithm designers to work on productive algorithms. For example, we can stop searching for a polynomial time algorithm that computes an exact solution to the problem. Instead, we might look for a *heuristic* algorithm that runs in polynomial time and computes a solution that is "close" to the exact one. The *quality* of a heuristic algorithm is usually evaluated and compared through empirical experiments [47]. Alternatively, we might design an approximation algorithm that extends the heuristic algorithm with the worst case guarantees on the quality of the solution produced by the algorithm [5].

Further detail about the theory of **NP**-completeness and approximation algorithms can be found in [5, 42, 91], for example. In Chapter 5 we will use the theory of **NP**-completeness to prove the complexity results for some computational problems in role-based access control, and design a number of heuristic algorithms to solve one of those problems in Chapter 6.

Chapter 3

The Oriented Permission RBAC Model and its Applications

Role-based access control and role hierarchies have generated considerable research activity in recent years. Crampton [29] suggested a role-based access control model that adopts a similar approach to RBAC96 with respect to the role hierarchy and the user-role assignment relation, but proposed a new approach to permissions and permission inheritance within the role hierarchy. In particular, permissions are 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. The motivation for this model is to address the deficiencies of the standard RBAC approach to inheritance and to offer certain advantages over existing role-based models. Hereafter we refer to this model as OP-RBAC (Oriented Permission RBAC).

The main contribution of this chapter is to investigate various applications of OP-RBAC. Since the introduction of RBAC, several authors have discussed the relationship between RBAC and the Bell-LaPadula model [69, 72, 73, 81]. Osborn *et al* [73] show that information flow policies in a number of different Bell-LaPadula models 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 these mandatory policies within the context of RBAC. We believe that this new approach is simpler, more natural, and more flexible 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 activate any role that is senior to any pair of roles that appear in a dynamic separation of duty constraint. Existing work, such as ERBAC96, requires a distinction to be made between role activation and permission usage hierarchies to solve this impasse. We will show how to use OP-RBAC to solve this problem without having to use a second hierarchy.

It has been shown that there are situations where it is useful to distinguish between role activation and permission usage inheritance [82]. Such a distinction has been made in both the ERBAC96 model [82] and the GTRBAC model [58], by introducing distinct role hierarchies. The final contribution of this chapter is to prove that any instance of the ERBAC96 model can be transformed into an instance of the OP-RBAC model, which requires a single role hierarchy.

The remainder of this chapter is organized as follows. In the next section, we formally present OP-RBAC and inter-relationships among the different components of the model. In Section 3.2, we briefly summarize the Bell-LaPadula model and introduce its extensions. In Section 3.3, we illustrate how OP-RBAC can be used to implement a number of different Bell-LaPadula models with the addition and modification of a few constraints to the basic model. We also discuss related work in this area and compare it to our approach. In Section 3.4, we consider two other applications of OP-RBAC: one

is to show that dynamic separation of duty constraints can be defined *and enforced* in a hierarchical RBAC model; the other is to demonstrate how to implement the ERBAC06 model using OP-RBAC. A preliminary version of this chapter appeared in 2007 [20].

3.1 The OP-RBAC model

The OP-RBAC model is a role-based access control model that contains a novel approach to permission inheritance. As we shall see, this model offers certain advantages over existing standard role-based models. We now formally introduce the characteristic features of OP-RBAC.

We assume the existence of a partially ordered set of roles $(R, \leqslant)$, a set of users U and a user-role assignment relation $UA \subseteq U \times R$. These components are identical to the ones in RBAC_1. We also assume the existence of a set of permissions P and define the permission-role assignment relation $PA \subseteq P \times R$.

The distinctive feature in OP-RBAC is that 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 $\mathsf{Roles}(p)$ and the set of roles authorized for p by the function $\mathsf{Roles}_E : P \to 2^R$, where

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

We say that $\mathsf{Roles}_E(p)$ is the (set of) *effective* roles for p.

Given a session s in which a set of roles activated $\mathsf{Roles}(s) \subseteq \downarrow\mathsf{Roles}(u)$, where $u = \mathsf{User}(s)$, a request by user u to invoke permission p in session s is only granted if u has activated one of p's effective roles in s, that is $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. Hence, RBAC_1 is a special case of OP-RBAC in which all permissions are up permissions, that is, $\mathsf{Roles}_E(p) = \uparrow\mathsf{Roles}(p)$ for all p.

3.2 The Bell-LaPadula model and extensions

The Bell-LaPadula model (BLP) is probably the most widely known security model and implements an information flow policy designed to preserve the confidentiality of information. To meet security requirements in different contexts, several versions of BLP have been developed, which differ in the use of security functions for labelling entities, and the definitions of the simple security property and the *-property. We assume the existence of a set of security functions Λ. A particular version of BLP chooses a subset of Λ for assigning security levels to subjects and objects, and defines the simple security property and the *-property with respect to the chosen security functions.

Table 3.1 summarizes the various BLP models in the literature, where π^{ss} denotes the simple security property and π^* denotes the *-property. We also write π_i^{ss} to denote the simple security property in BLP_i, and π_i^* to denote the *-property in BLP_i. Note that all BLP models include the discretionary security property, which we denote by π^{ds}. It requires that all requests are authorized by the protection matrix M. Recall that V models the authorizations that have been granted and not yet revoked by the system. BLP_0 corresponds to the simple version of BLP introduced in Chapter 2. Note that as a consequence of π_0^{ss} and π_0^*, a subject s is authorized to write (that is, simultaneously read and append access) an object o only if $\lambda(s) = \lambda(o)$. BLP_1

introduces the strict *-property ($\lambda(s) = \lambda(o)$ for append access) that is used to prevent integrity or covert channel problems due to write up [81]. In order to give subjects more power than allowed by π_0^{ss} and π_0^*, BLP_2 associates a pair of security labels $\lambda_r(s)$ and $\lambda_a(s)$ with each subject and requires that $\lambda_r(s) \geqslant \lambda_a(s)$. The simple security property is applied with respect to $\lambda_r(s)$ and the *-property to $\lambda_a(s)$. Consequently, any subject s is allowed to write an object whose security label is in the *trust range* of labels between $\lambda_r(s)$ and $\lambda_a(s)$ [9]. It can be seen that the information flow policy implemented in BLP_2 is partly relaxed to achieve selective downgrading of information from a high security level to a low security level.

Model	Λ	π^{ss}	π^*
BLP_0	$\lambda : S \cup O \to L$	$\forall(s, o, \texttt{read}) \in V,$ $\lambda(s) \geqslant \lambda(o)$	$\forall(s, o, \texttt{append}) \in V,$ $\lambda(s) \leqslant \lambda(o)$
BLP_1	$\lambda : S \cup O \to L$	$\forall(s, o, \texttt{read}) \in V,$ $\lambda(s) \geqslant \lambda(o)$	$\forall(s, o, \texttt{append}) \in V,$ $\lambda(s) = \lambda(o)$
BLP_2	$\lambda : O \to L$ $\lambda_r : S \to L$ $\lambda_a : S \to L$ $\forall s \in S,\ \lambda_a(s) \leqslant \lambda_r(s)$	$\forall(s, o, \texttt{read}) \in V,$ $\lambda_r(s) \geqslant \lambda(o)$	$\forall(s, o, \texttt{append}) \in V,$ $\lambda_a(s) \leqslant \lambda(o)$
BLP_3	$S_T \subseteq S$ $S' = S \setminus S_T$ $\lambda : S \cup O \to L$ $\lambda_c : S \to L$ $\forall s \in S,\ \lambda_c(s) \leqslant \lambda(s)$	$\forall(s, o, \texttt{read}) \in V,$ $\lambda(s) \geqslant \lambda(o)$	$\forall(s', o, \texttt{read}) \in V,$ $\lambda_c(s') \geqslant \lambda(o)$ $\forall(s', o, \texttt{append}) \in V,$ $\lambda_c(s') \leqslant \lambda(o)$ $\forall(s', o, \texttt{write}) \in V,$ $\lambda_c(s') = \lambda(o)$

Table 3.1: A summary of the different Bell-LaPadula models

BLP_3 is a modified Bell-LaPadula model [8, 12] that introduces two important concepts: *current security label* of subjects and *trusted subjects*. When a user u logs in to a computer system, a subject s is created and operates at a current security label $\lambda_c(s)$, where $\lambda_c(s) \leqslant \lambda(u)$. We define $\lambda(u) = \lambda(s)$, which means the clearance level of u is recorded as the *maximum* level of s. The *-property (π_3^*) is defined in terms of the current security label of subjects for read, append and write access. As an immediate

consequence of π_3^*, if a subject is simultaneously reading and appending to two different objects, the object o that is being appended to will have at least as high a security label as the object o' that is being read.

Trusted subjects S_T are those subjects not constrained by π_3^*. In other words, a trusted subject is one assumed not to copy high level information into a low level object even if it is possible [12]. Hence, π_3^* is applied only to untrusted subjects.

On the other hand, the simple security property (π_3^{ss}) is applied to (trusted and untrusted) subjects S, and is defined with respect to $\lambda(s)$ for read access. It is worth noting that given a set of untrusted subjects S', the satisfaction of π_3^* with respect to S' implies the satisfaction of π_3^{ss} with respect to S' [66]. This is because if π_3^* is satisfied with respect to $s' \in S'$ then we have $\lambda_c(s') \geqslant \lambda(o)$ for read access, and by definition $\lambda(s') \geqslant \lambda_c(s')$.

3.3 Implementing BLP using OP-RBAC

We now demonstrate how, with the addition and subsequent modification of a few constraints, OP-RBAC can be used to implement the BLP models described in Section 3.2. The most important contribution is that OP-RBAC provides a more direct implementation of the BLP models than has previously been possible using role-based models [72, 73]. In addition, OP-RBAC supports the assignment of compound permissions (write permissions) which include both read and append access to objects.

We define the state of an OP-RBAC system to be a tuple $(W, (UA, RH, PA))$. The set (UA, RH, PA) models the access control state, which is used to compute those access requests that are authorized and would be granted by the access control mechanism. The set $W \subseteq U \times P$ models the set of permissions that are currently "active"; that is, those permissions that have been granted to users by the access control mechanism.

Initially we ignore π^{ds} as it is common to all BLP models. In Section 3.3.7 we discuss how limited support for π^{ds} can be provided in OP-RBAC, unlike existing RBAC approaches [73], which do not consider this aspect of BLP at all.

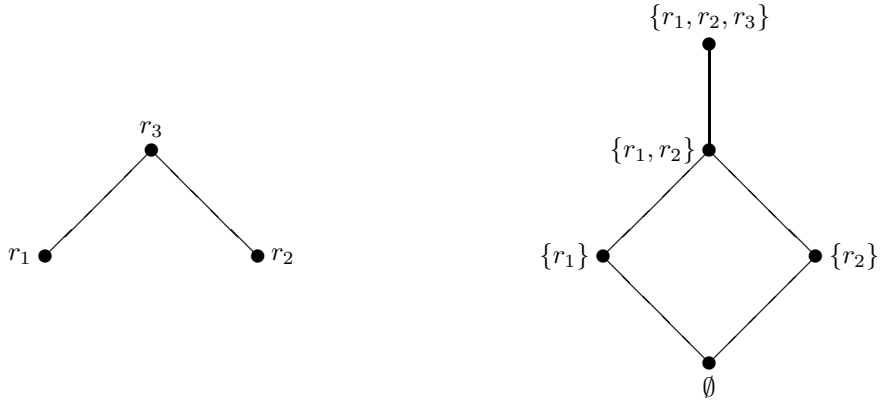
3.3.1 Security labels

Crampton [29] discussed why the set of roles assigned to a user can be interpreted as a security label for that user, while the set of roles assigned to a permission does not have the same interpretation. We now briefly describe this interpretation, which inspires us to consider how we might define suitable security labels for users, permissions and objects in a role-based model.

Let $(R, \leqslant)$ be a partially ordered set of roles and let $\mathsf{Roles}(u)$ denote the set of roles explicitly assigned to the user u. Let $\mathcal{L}(R) = \{\downarrow S : S \subseteq R\}$, then $(\mathcal{L}(R), \subseteq)$ is the lattice of order ideals in R.[1] We can regard $\downarrow\mathsf{Roles}(u)$ in $(\mathcal{L}(R), \subseteq)$ as the actual security label of u. For example, Figure 3.1 depicts a role hierarchy $(R, \leqslant)$ and the associated lattice $(\mathcal{L}(R), \subseteq)$. If $\mathsf{Roles}(u) = \{r_3\}$, then $\downarrow\mathsf{Roles}(u) = \{r_1, r_2, r_3\}$ can be regarded as a security label of u. A user u can activate a session s using some subset of the roles authorized for u, that is $\mathsf{Roles}(s) \subseteq \downarrow\mathsf{Roles}(u)$. We can regard the label $\downarrow\mathsf{Roles}(s)$ in $(\mathcal{L}(R), \subseteq)$ as the current security label of u. For example, a user assigned to r_3 can create a session using role r_1, thereby creating a current security label of $\{r_1\}$. In short, 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.[2] In other words, the user's security label is system-defined and the current security label is chosen by the user, which corresponds closely to λ and λ_c in BLP_3.

[1] If X is a poset then $Y \subseteq X$ is an order ideal if for all $y \in Y$, $x \in X$, $x \leqslant y$ implies $x \in Y$.

[2] In actual fact, we associate the sets $\mathsf{Roles}(u)$ and $\mathsf{Roles}(s)$ with the respective security labels $\downarrow\mathsf{Roles}(u)$ and $\downarrow\mathsf{Roles}(s)$ in $\mathcal{L}(R)$.



(a) $\langle R, \leqslant \rangle$ (b) $\langle \mathcal{L}(R), \subseteq \rangle$

Figure 3.1: A role hierarchy and the associated lattice

However, we can not define the security label of a permission in terms of the set of roles explicitly assigned to the 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 is authorized for 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 is authorized for permission p (to read an object) if the security label of u dominates the security label of p. In other words, if we interpret the security label of a permission to be $\downarrow\mathsf{Roles}(p)$, we would require that $\downarrow\mathsf{Roles}(p) \subseteq \downarrow\mathsf{Roles}(u)$. Consider the following situation: $\mathsf{Roles}(u) = \{r_1\}$ and $\mathsf{Roles}(p) = \{r_1, r_2\}$. In RBAC, u is authorized for p, because there exists a role r_1 that both u and p are assigned to. However, in BLP, u is not able to perform p, because $\downarrow\mathsf{Roles}(u) = \{r_1\} \subseteq \downarrow\mathsf{Roles}(p) = \{r_1, r_2\}$.

The incompatibility can be resolved by assigning each permission p to a unique role r. That is $\mathsf{Roles}(p) = \{r\}$ for some $r \in R$ (as is assumed in existing approaches). In this case, the security label of the permission is $\downarrow r$. Moreover, we require that all permissions for a particular object be assigned to the same role r, thereby the security label of the object is $\downarrow r$.

The limitation of this approach is that we need to construct an additional lattice of

security labels based on the partially ordered set of roles. For each user and permission, we are also required to map the set of roles associated with the user or the permission to a security label in the lattice.

An alternative approach is to introduce more restrictive user-role assignment and permission-role assignment relations in OP-RBAC. We assume that user-role assignment and permission-role assignment are functions, that is, for all $u \in U$ and $p \in P$, $\mathsf{Roles}(u) = r$ and $\mathsf{Roles}(p) = r'$ for some $r, r' \in R$. Hence, roles r and r' can be interpreted as the security labels for user u and permission p respectively. Note that this approach of defining security labels of a user and of a permission is adopted by most existing work [73, 81], because the simplicity of this approach provides a natural implementation of BLP models, although it would be a significant restriction in general RBAC models. Along with this approach, we now demonstrate how OP-RBAC can be constrained to simulate BLP_0, BLP_1, BLP_2, and BLP_3.

3.3.2 $\mathbf{BLP}_0$

Recall that a BLP_0 system is defined by $(L, \leqslant)$ and $\lambda : U \cup O \to L$. Granted requests must satisfy π_0^{ss} and π_0^*. In order to implement BLP_0 using OP-RBAC, we set $(R, \leqslant)$ equal to $(L, \leqslant)$. In addition, we define the following constraints:

Constraint 3.3.1 *UA is a function; in other words, each user is assigned to a unique role. The* security label *of u is defined to be r, where $(u, r) \in UA$.*

Constraint 3.3.2 *A user u has no choice when running a session; the unique role to which he is assigned is activated.*

Constraint 3.3.3 *PA is a function; each permission is assigned to a unique role.*

Constraint 3.3.4 *Each permission has the form (o, m), where $m \in$*

$\{\texttt{read}, \texttt{append}, \texttt{write}\}$ *and* o *is an object. We require that all three permissions are assigned to the same role* r*. The* security label *of* o *is defined to be* r*.*

Constraint 3.3.5 *If* $p = (o, \texttt{read})$ *then* $p \in P^+$*.*

Constraint 3.3.6 *If* $p = (o, \texttt{append})$ *then* $p \in P^-$*.*

Constraint 3.3.7 *If* $p = (o, \texttt{write})$ *then* $p \in P^0$*.*

We now prove that given a BLP_0 system, we have an OP-RBAC system with appropriate constraints defined above that is equivalent to the BLP_0 system, in the sense that the same set of requests is authorized.

Theorem 3.3.1 *Given a* BLP_0 *system* Σ *defined by* $(L, \leqslant)$ *and* λ*, and an OP-RBAC system* Σ' *that satisfies constraints* 3.3.1–3.3.7*, then a request* (u, p) *is authorized in* Σ *(by satisfying* π_0^{ss} *and* π_0^**) if and only if* (u, p) *is authorized in* Σ'*.*

Proof We first prove the "if" condition.

Let $W \subseteq U \times P$ be the set of permissions that have been granted to users in the OP-RBAC system Σ'. Let $(u, p) \in W$, where $p = (o, \texttt{read})$, $(p, r) \in PA$, and $(u, r') \in UA$. By Constraint 3.3.5, we have $\mathsf{Roles}_E(p) = \uparrow r$. Moveover, since $(u, p) \in W$, there exists a session s activated by u such that $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. By Constraint 3.3.2, $\mathsf{Roles}(s) = \{r'\}$. Hence $r' \in \uparrow r$; that is $r' \geqslant r$. By Constraint 3.3.1, $\lambda(u) = r'$, and by Constraint 3.3.4, $\lambda(o) = r$; that is $\lambda(u) \geqslant \lambda(o)$. Hence π_0^{ss} is satisfied.

Let $(u, p) \in W$, where $p = (o, \texttt{append})$, $(p, r) \in PA$, and $(u, r') \in UA$. By Constraint 3.3.6, we have $\mathsf{Roles}_E(p) = \downarrow r$. Moveover, since $(u, p) \in W$, there exists a session s activated by u such that $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. By Constraint 3.3.2, $\mathsf{Roles}(s) = \{r'\}$. Hence $r' \in \downarrow r$; that is $r' \leqslant r$. By Constraint 3.3.1, $\lambda(u) = r'$, and by Constraint 3.3.4, $\lambda(o) = r$; that is $\lambda(u) \leqslant \lambda(o)$. Hence π_0^* is satisfied.

Let $(u, p) \in W$, where $p = (o, \texttt{write})$, $(p, r) \in PA$, and $(u, r') \in UA$. By Constraint 3.3.7, we have $\mathsf{Roles}_E(p) = \{r\}$. Moveover, since $(u, p) \in W$, there exists a session s activated by u such that $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. By Constraint 3.3.2, $\mathsf{Roles}(s) = \{r'\}$. Hence $r' = r$. By Constraint 3.3.1, $\lambda(u) = r'$, and by Constraint 3.3.4, $\lambda(o) = r$; that is $\lambda(u) = \lambda(o)$. Hence π_0^{ss} and π_0^* are satisfied.

We now prove the "only if" condition.

Let V be the set of requests that have been granted in the BLP_0 system Σ. Let $(u, p) \in V$, where $p = (o, \texttt{read})$. By π_0^{ss}, $\lambda(u) \geqslant \lambda(o)$. By Constraint 3.3.1, $\lambda(u) = \mathsf{Roles}(u) = r'$. By Constraint 3.3.2, u can only activate r' in a session s, that is $\mathsf{Roles}(s) = r'$. By Constraint 3.3.4, $\lambda(o) = \mathsf{Roles}(p) = r$, and by Constraint 3.3.5, $\mathsf{Roles}_E(p) = \uparrow\mathsf{Roles}(p) = \uparrow r$. Since $\lambda(u) \geqslant \lambda(o)$, $r' \in \uparrow r$. Hence $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. Therefore u is authorized for p in Σ'.

Let $(u, p) \in V$, where $p = (o, \texttt{append})$. By π_0^*, $\lambda(u) \leqslant \lambda(o)$. By Constraint 3.3.1, $\lambda(u) = \mathsf{Roles}(u) = r'$. By Constraint 3.3.2, u can only activate r' in a session s, that is $\mathsf{Roles}(s) = r'$. By Constraint 3.3.4, $\lambda(o) = \mathsf{Roles}(p) = r$, and by Constraint 3.3.6, $\mathsf{Roles}_E(p) = \downarrow\mathsf{Roles}(p) = \downarrow r$. Since $\lambda(u) \leqslant \lambda(o)$, $r' \in \downarrow r$. Hence $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. Therefore u is authorized for p in Σ'.

Let $(u, p) \in V$, where $p = (o, \texttt{write})$. By π_0^{ss} and π_0^*, $\lambda(u) = \lambda(o)$. By Constraint 3.3.1, $\lambda(u) = \mathsf{Roles}(u) = r'$. By Constraint 3.3.2, u can only activate r' in a session s, that is $\mathsf{Roles}(s) = r'$. By Constraint 3.3.4, $\lambda(o) = \mathsf{Roles}(p) = r$, and by Constraint 3.3.7, $\mathsf{Roles}_E(p) = \mathsf{Roles}(p) = r$. Since $\lambda(u) = \lambda(o)$, $r' = r$. Hence $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. Therefore u is authorized for p in Σ'. ■

Remark 3.3.1 *Previous work in this area can not deal with write permissions [73, 81], something we believe detracts significantly from the utility of existing approaches. An important consequence of Theorem 3.3.1 is that for any write permission to be granted,*

the user must have exactly the same security label as the object. In other words, our approach deals with write permissions appropriately, and in the manner intended by the original BLP model. For this reason alone, we believe our approach is to be preferred to existing work in the literature.

3.3.3 $\mathbf{BLP}_1$

The use of the strict *-property in BLP_1 requires a simple modification to Constraint 3.3.6.

Constraint 3.3.8 *If* $p = (o, \texttt{append})$ *then* $p \in P^0$.

By Constraint 3.3.8, a user is allowed to perform an append permission only if the user has activated the role to which the append permission is assigned. This is analogous to the strict *-property in BLP_1 where a user must have the same security label as an object in order to append to it.

Theorem 3.3.2 *Given a BLP*$_1$ *system* Σ *defined by* $(L, \leqslant)$ *and* λ*, and an OP-RBAC system* Σ' *that satisfies constraints* 3.3.1–3.3.5 *and* 3.3.7–3.3.8*, then a request* (u, p) *is authorized in* Σ *(by satisfying* π_1^{ss} *and* π_1^**) if and only if* (u, p) *is authorized in* Σ'.

The proof of this result is similar to that of Theorem 3.3.1 and is omitted.

3.3.4 $\mathbf{BLP}_2$

In the trusted network interpretation of multi-level security (BLP_2), each user is associated with a pair of security labels $\lambda_r(u)$ and $\lambda_a(u)$, where $\lambda_a(u) \leqslant \lambda_r(u)$. The simple security property π_2^{ss} is applied with respect to $\lambda_r(u)$ and the *-property π_2^* with respect to $\lambda_a(u)$. As a consequence, the user can invoke a write permission on an object whose security label is contained in the range $[\lambda_a(u), \lambda_r(u)]$. Clearly, BLP_0 is a

subcase of BLP_2, where $\lambda_r(u) = \lambda_a(u)$. To implement BLP_2 in OP-RBAC, we replace Constraints 3.3.1 and 3.3.2 with the following constraints.

Constraint 3.3.9 *Each user u is assigned to at most two roles r_a and r_r with $r_a \leqslant r_r$. That is $\mathsf{Roles}(u) = \{r_a, r_r\}$, where r_r corresponds to $\lambda_r(u)$ and r_a corresponds to $\lambda_a(u)$.*

Constraint 3.3.10 *A user assigned to roles r_a and r_r has no choice when running a session; the roles in $[r_a, r_r]$ are activated.*

Theorem 3.3.3 *Given a BLP_2 system Σ defined by $(L, \leqslant)$, λ_r, λ_a and λ, and an OP-RBAC system Σ' that satisfies constraints* 3.3.3–3.3.7 *and* 3.3.9–3.3.10*, then a request (u, p) is authorized in Σ (by satisfying π_2^{ss} and π_2^*) if and only if (u, p) is authorized in Σ'.*

Proof We first prove the "if" condition.

Let $(u, p) \in W$, where $p = (o, \texttt{read})$, $(p, r) \in PA$, $(u, r_r) \in UA$, and $(u, r_a) \in UA$. By Constraint 3.3.5, we have $\mathsf{Roles}_E(p) = \uparrow r$. Moveover, since $(u, p) \in W$, there exits a session s activated by u such that $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$; that is, there exists $r' \in \mathsf{Roles}(s)$ such that $r' \geqslant r$. By Constraint 3.3.10, $r_r \geqslant r' \geqslant r_a$; that is $r_r \geqslant r$. By Constraint 3.3.9, $\lambda_r(u) = r_r$, and by Constraint 3.3.4, $\lambda(o) = r$; that is $\lambda_r(u) \geqslant \lambda(o)$. Hence π_2^{ss} is satisfied.

Let $(u, p) \in W$, where $p = (o, \texttt{append})$, $(p, r) \in PA$, $(u, r_r) \in UA$, and $(u, r_a) \in UA$. By Constraint 3.3.6, we have $\mathsf{Roles}_E(p) = \downarrow r$. Moveover, since $(u, p) \in W$, there exits a session s activated by u such that $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$; that is, there exists $r' \in \mathsf{Roles}(s)$ such that $r' \leqslant r$. By Constraint 3.3.10, $r_a \leqslant r' \leqslant r_r$; that is $r_a \leqslant r$. By Constraint 3.3.9, $\lambda_a(u) = r_a$, and by Constraint 3.3.4, $\lambda(o) = r$; that is $\lambda_a(u) \leqslant \lambda(o)$. Hence π_2^* is satisfied.

Let $(u, p) \in W$, where $p = (o, \texttt{write})$, $(p, r) \in PA$, $(u, r_r) \in UA$, and $(u, r_a) \in UA$. By Constraint 3.3.7, we have $\mathsf{Roles}_E(p) = \{r\}$. Moveover, since $(u, p) \in W$, there exits

a session s activated by u such that $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$; that is, there exists $r' \in \mathsf{Roles}(s)$ such that $r' = r$. By Constraint 3.3.10, $r_a \leqslant r' \leqslant r_r$, by Constraint 3.3.9, $\lambda_r(u) = r_r$ and $\lambda_a(u) = r_a$, and by Constraint 3.3.4, $\lambda(o) = r$; that is $\lambda_a(u) \leqslant \lambda(o) \leqslant \lambda_r(u)$. Hence π_2^{ss} and π_2^* are satisfied.

We now prove the "only if" condition.

Let V be the set of requests that have been granted in the BLP_2 system Σ. Let $(u, p) \in V$, where $p = (o, \texttt{read})$. By π_2^{ss}, $\lambda_r(u) \geqslant \lambda(o)$. By Constraint 3.3.9, $\lambda_r(u) = r_r \in \mathsf{Roles}(u)$. By Constraint 3.3.10, u must activate a set of roles including r_r in a session s, that is $r_r \in \mathsf{Roles}(s)$. By Constraint 3.3.4, $\lambda(o) = \mathsf{Roles}(p) = r$, and by Constraint 3.3.5, $\mathsf{Roles}_E(p) = \uparrow\mathsf{Roles}(p) = \uparrow r$. Since $\lambda_r(u) \geqslant \lambda(o)$, $r_r \in \uparrow r$. Hence $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. Therefore u is authorized for p in Σ'.

Let $(u, p) \in V$, where $p = (o, \texttt{append})$. By π_2^*, $\lambda_a(u) \leqslant \lambda(o)$. By Constraint 3.3.9, $\lambda_a(u) = r_a \in \mathsf{Roles}(u)$. By Constraint 3.3.10, u must activate a set of roles including r_a in a session s, that is $r_a \in \mathsf{Roles}(s)$. By Constraint 3.3.4, $\lambda(o) = \mathsf{Roles}(p) = r$, and by Constraint 3.3.6, $\mathsf{Roles}_E(p) = \downarrow\mathsf{Roles}(p) = \downarrow r$. Since $\lambda_a(u) \leqslant \lambda(o)$, $r_a \in \downarrow r$. Hence $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. Therefore u is authorized for p in Σ'.

Let $(u, p) \in V$, where $p = (o, \texttt{write})$. By π_2^{ss} and π_2^*, $\lambda_a(u) \leqslant \lambda(o) \leqslant \lambda_r(u)$. By Constraint 3.3.9, $\mathsf{Roles}(u) = \{r_r, r_a\}$, where $r_r = \lambda_r(u)$ and $r_a = \lambda_a(u)$. By Constraint 3.3.10, u must activate all roles $[r_a, r_r]$ in a session s, that is $\mathsf{Roles}(s) = [r_a, r_r]$. By Constraint 3.3.4, $\lambda(o) = \mathsf{Roles}(p) = r$, and by Constraint 3.3.7, $\mathsf{Roles}_E(p) = \mathsf{Roles}(p) = r$. Since $\lambda_a(u) \leqslant \lambda(o) \leqslant \lambda_r(u)$, $r \in [r_a, r_r]$. Hence $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. Therefore u is authorized for p in Σ'. ■

3.3.5 BLP_3

BLP_3 introduces the current security function λ_c to simplify the evaluation of the *-property π_3^*[3], and to improve usability by allowing a user to downgrade his security level. In addition, BLP_3 introduces trusted subjects that are allowed to operate without the extra encumbrance of π_3^*. Typical examples of trusted subjects in the context of operating systems are device drivers and memory management software.

In order to provide close correlations between BLP_3 and OP-RBAC, from now on, we assume that no subject in BLP_3 could be trusted not to copy high level information into low level objects. In other words, all subjects in BLP_3 are untrusted subjects who subject to π_3^{ss} and π_3^*. In this case, as we discussed in Section 3.2, π_3^* implies π_3^{ss}. Hence, a request from a subject to access an object is granted in a BLP_3 system only if this request satisfies π_3^*. In order to implement BLP_3 in OP-RBAC, we replace Constraint 3.3.2 with the following constraint.

Constraint 3.3.11 *Each user u can only activate a single role $r' \in \downarrow\mathsf{Roles}(u) = \downarrow r$ when running a session s. The current security label of u is defined to be $\mathsf{Roles}(s) = \{r'\}$ for some $r' \leqslant r$.*

Theorem 3.3.4 *Given a BLP_3 system Σ defined by $(L, \leqslant)$, λ_c, and λ, and an OP-RBAC system Σ' that satisfies constraints* 3.3.1, 3.3.3–3.3.7 *and* 3.3.11*, then a request (u, p) is authorized in Σ (by satisfying π_3^*) if and only if (u, p) is authorized in Σ'.*

Proof We first prove the "if" condition.

Let $(u, p) \in W$, where $p = (o, \texttt{read})$, $(p, r) \in PA$, and $(u, r') \in UA$. By Constraint 3.3.5, we have $\mathsf{Roles}_E(p) = \uparrow r$. Moveover, since $(u, p) \in W$, there exists a session s activated by u such that $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. By Constraint 3.3.11,

[3]The *-property in the earlier formulation of the Bell-LaPadula model [10, 11] requires that the decision whether to grant a request from a subject to write an object, for example, can only be made by considering the security labels of all objects the subject has previously read.

$\mathsf{Roles}(s) = \{r''\}$. Hence $r'' \in \uparrow r$; that is $r'' \geqslant r$. By Constraint 3.3.11, $\lambda_c(u) = r''$, and by Constraint 3.3.4, $\lambda(o) = r$; that is $\lambda_c(u) \geqslant \lambda(o)$. By Constraint 3.3.1, $\lambda(u) = r'$, and by Constraint 3.3.11, $r'' \leqslant r'$; that is $\lambda_c(u) \leqslant \lambda(u)$. Hence the first part of π_3^* is satisfied.

Let $(u, p) \in W$, where $p = (o, \texttt{append})$, $(p, r) \in PA$, and $(u, r') \in UA$. By Constraint 3.3.6, we have $\mathsf{Roles}_E(p) = \downarrow r$. Moveover, since $(u, p) \in W$, there exists a session s activated by u such that $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. By Constraint 3.3.11, $\mathsf{Roles}(s) = \{r''\}$. Hence $r'' \in \downarrow r$; that is $r'' \leqslant r$. By Constraint 3.3.11, $\lambda_c(u) = r''$, and by Constraint 3.3.4, $\lambda(o) = r$; that is $\lambda_c(u) \leqslant \lambda(o)$. Obviously, $r'' \leqslant r'$, hence $\lambda_c(u) \leqslant \lambda(u)$. Hence the second part of π_3^* is satisfied.

Let $(u, p) \in W$, where $p = (o, \texttt{write})$, $(p, r) \in PA$, and $(u, r') \in UA$. By Constraint 3.3.7, we have $\mathsf{Roles}_E(p) = \{r\}$. Moveover, since $(u, p) \in W$, there exists a session s activated by u such that $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. By Constraint 3.3.11, $\mathsf{Roles}(s) = \{r''\}$. Hence $r'' = r$. By Constraint 3.3.11, $\lambda_c(u) = r''$, and by Constraint 3.3.4, $\lambda(o) = r$; that is $\lambda_c(u) = \lambda(o)$. Obviously, $r'' \leqslant r'$, hence $\lambda_c(u) \leqslant \lambda(u)$. Hence the third part of π_3^* is satisfied.

We now prove the "only if" condition.

Let V be the set of requests that have been granted in the BLP_3 system Σ. Let $(u, p) \in V$, where $p = (o, \texttt{read})$. By π_3^*, $\lambda(o) \leqslant \lambda_c(u)$, and $\lambda_c(u) \leqslant \lambda(u)$. By Constraint 3.3.1, $\lambda(u) = \mathsf{Roles}(u) = r'$. By Constraint 3.3.11, u can only activate a role r'' in a session s, that is $\lambda_c(u) = \mathsf{Roles}(s) = r''$, where $r'' \leqslant r'$ corresponds to $\lambda_c(u) \leqslant \lambda(u)$. By Constraint 3.3.4, $\lambda(o) = \mathsf{Roles}(p) = r$, and by Constraint 3.3.5, $\mathsf{Roles}_E(p) = \uparrow\mathsf{Roles}(p) = \uparrow r$. Since $\lambda_c(u) \geqslant \lambda(o)$, $r'' \in \uparrow r$. Hence $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. Therefore u is authorized for p in Σ'.

Let $(u, p) \in V$, where $p = (o, \texttt{append})$. By π_3^*, $\lambda_c(u) \leqslant \lambda(o)$, and $\lambda_c(u) \leqslant \lambda(u)$. By Constraint 3.3.1, $\lambda(u) = \mathsf{Roles}(u) = r'$. By Constraint 3.3.11, u can only activate

a role r'' in a session s, that is $\lambda_c(u) = \mathsf{Roles}(s) = r''$, where $r'' \leqslant r'$ corresponds to $\lambda_c(u) \leqslant \lambda(u)$. By Constraint 3.3.4, $\lambda(o) = \mathsf{Roles}(p) = r$, and by Constraint 3.3.6, $\mathsf{Roles}_E(p) = \downarrow\mathsf{Roles}(p) = \downarrow r$. Since $\lambda_c(u) \leqslant \lambda(o)$, $r'' \in \downarrow r$. Hence $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. Therefore u is authorized for p in Σ'.

Let $(u, p) \in V$, where $p = (o, \texttt{write})$. By π_3^*, $\lambda_c(u) = \lambda(o)$ and $\lambda_c(u) \leqslant \lambda(u)$. By Constraint 3.3.1, $\lambda(u) = \mathsf{Roles}(u) = r'$. By Constraint 3.3.11, u can only activate a role r'' in a session s, that is $\lambda_c(u) = \mathsf{Roles}(s) = r''$, where $r'' \leqslant r'$ corresponds to $\lambda_c(u) \leqslant \lambda(u)$. By Constraint 3.3.4, $\lambda(o) = \mathsf{Roles}(p) = r$, and by Constraint 3.3.7, $\mathsf{Roles}_E(p) = \mathsf{Roles}(p) = r$. Since $\lambda_c(u) = \lambda(o)$, $r'' = r$. Hence $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. Therefore u is authorized for p in Σ'. ■

3.3.6 $\mathbf{BLP}_4$

A further extension of BLP_0, which we call BLP_4, associates each object with two different security labels, $\lambda_r(o)$ and $\lambda_a(o)$; π_4^{ss} is defined using $\lambda_r(o)$ and π_4^* is defined using $\lambda_a(o)$. We might require, by analogy with BLP_2, that the "append-level" of an object is higher than the "read-level"; that is $\lambda_a(o) \geqslant \lambda_r(o)$. In this case, a subject whose security label is contained in the range $[\lambda_r(o), \lambda_a(o)]$ can have write access to the object. This is the analogue of the trusted range of subjects in the trusted network interpretation (BLP_2).

However, note that we can support some useful access control policies if we drop the requirement that $\lambda_a(o) \geqslant \lambda_r(o)$, in which case no user can both read and append (write) that object o. Consider an object such as an audit log file f: we require that low level users must append to f, while high level users can read f but must not to be able change it. We can implement these requirements simply by making $\lambda_a(f) < \lambda_r(f)$. Hence, in BLP_4, we do not insist that $\lambda_r(o) \leqslant \lambda_a(o)$. Table 3.2 summarizes BLP_4.

To implement BLP_4 in OP-RBAC, we replace Constraints 3.3.3 and 3.3.4 with the

Model	Λ	π^{ss}	π^*
BLP_4	$\lambda : S \to L$ $\lambda_r : O \to L$ $\lambda_a : O \to L$	$\forall (s, o, \texttt{read}) \in V$, $\lambda(s) \geqslant \lambda_r(o)$	$\forall (s, o, \texttt{append}) \in V$, $\lambda(s) \leqslant \lambda_a(o)$

Table 3.2: BLP_4

following constraint.

Constraint 3.3.12 *If $p = (o, m)$, where $m \in \{\texttt{read}, \texttt{append}\}$, p is assigned to a unique role, but permissions for the same object, $(o, \texttt{read})$ and $(o, \texttt{append})$, may be assigned to different roles. The assignment of a write permission for the same object, $(o, \texttt{write})$, depends on the ordering between roles to which $(o, \texttt{read})$ and $(o, \texttt{append})$ are assigned. More specifically, for each object o such that $((o, \texttt{read}), r_r), ((o, \texttt{append}), r_a) \in PA$, the permission $(o, \texttt{write})$ should assigned to all roles in $[r_r, r_a]$ if $r_r \leqslant r_a$. In the case that $r_r \nleqslant r_a$, this permission can not be assigned to any role. Hence, r_r corresponds to $\lambda_r(o)$ and r_a corresponds to $\lambda_a(o)$.*

Theorem 3.3.5 *Given a BLP_4 system Σ defined by $(L, \leqslant)$, λ_r, λ_a and λ, and an OP-RBAC system Σ' that satisfies constraints* 3.3.1*,* 3.3.2*,* 3.3.5–3.3.7*, and* 3.3.12*, then a request (u, p) is authorized in Σ (by satisfying π_4^{ss} and π_4^*) if and only if (u, p) is authorized in Σ'.*

Proof We first prove the "if" condition.

Let $(u, p) \in W$, where $p = (o, \texttt{read})$, $(p, r_r) \in PA$, and $(u, r) \in UA$. By Constraint 3.3.5, we have $\mathsf{Roles}_E(p) = \uparrow r_r$. Moveover, since $(u, p) \in W$, there exists a session s activated by u such that $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. By Constraint 3.3.2, $\mathsf{Roles}(s) = \{r\}$. Hence $r \in \uparrow r_r$; that is $r \geqslant r_r$. By Constraint 3.3.1, $\lambda(u) = r$, and by Constraint 3.3.12, $\lambda_r(o) = r_r$; that is $\lambda(u) \geqslant \lambda_r(o)$. Hence π_4^{ss} is satisfied.

Let $(u, p) \in W$, where $p = (o, \texttt{append})$, $(p, r_a) \in PA$, and $(u, r) \in UA$. By Constraint 3.3.6, we have $\mathsf{Roles}_E(p) = \downarrow r_a$. Moveover, since $(u, p) \in W$, there exists a

session s activated by u such that $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. By Constraint 3.3.2, $\mathsf{Roles}(s) = \{r\}$. Hence $r \in \downarrow r_a$; that is $r \leqslant r_a$. By Constraint 3.3.1, $\lambda(u) = r$, and by Constraint 3.3.12, $\lambda_a(o) = r_a$; that is $\lambda(u) \leqslant \lambda_a(o)$. Hence π_4^* is satisfied.

Let $(u, p) \in W$, where $p = (o, \texttt{write})$ and $(u, r) \in UA$. Let $((o, \texttt{read}), r_r), ((o, \texttt{append}), r_a) \in PA$. By Constraint 3.3.12, $\mathsf{Roles}(p) = [r_r, r_a]$. By Constraint 3.3.7, we have $\mathsf{Roles}_E(p) = \mathsf{Roles}(p) = [r_r, r_a]$. Moveover, since $(u, p) \in W$, there exists a session s activated by u such that $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. By Constraint 3.3.2, $\mathsf{Roles}(s) = \{r\}$. Hence $r \in [r_r, r_a]$. By Constraint 3.3.1, $\lambda(u) = r$, and by Constraint 3.3.12, $\lambda_r(o) = r_r$, $\lambda_a(o) = r_a$; that is $\lambda_r(o) \leqslant \lambda(u) \leqslant \lambda_a(o)$. Hence π_4^{ss} and π_4^* are satisfied.

We now prove the "only if" condition.

Let V be the set of requests that have been granted in the BLP_4 system Σ. Let $(u, p) \in V$, where $p = (o, \texttt{read})$. By π_4^{ss}, $\lambda(u) \geqslant \lambda_r(o)$. By Constraint 3.3.1, $\lambda(u) = \mathsf{Roles}(u) = r'$. By Constraint 3.3.2, u can only activate r' in a session s, that is $\mathsf{Roles}(s) = r'$. By Constraint 3.3.12, $\lambda_r(o) = \mathsf{Roles}(p) = r_r$, and by Constraint 3.3.5, $\mathsf{Roles}_E(p) = \uparrow\mathsf{Roles}(p) = \uparrow r_r$. Since $\lambda(u) \geqslant \lambda_r(o)$, $r' \in \uparrow r_r$. Hence $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. Therefore u is authorized for p in Σ'.

Let $(u, p) \in V$, where $p = (o, \texttt{append})$. By π_4^*, $\lambda(u) \leqslant \lambda_a(o)$. By Constraint 3.3.1, $\lambda(u) = \mathsf{Roles}(u) = r'$. By Constraint 3.3.2, u can only activate r' in a session s, that is $\mathsf{Roles}(s) = r'$. By Constraint 3.3.12, $\lambda_a(o) = \mathsf{Roles}(p) = r_a$, and by Constraint 3.3.6, $\mathsf{Roles}_E(p) = \downarrow\mathsf{Roles}(p) = \downarrow r_a$. Since $\lambda(u) \leqslant \lambda_a(o)$, $r' \in \downarrow r_a$. Hence $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. Therefore u is authorized for p in Σ'.

Let $(u, p) \in V$, where $p = (o, \texttt{write})$. By π_4^{ss} and π_4^*, $\lambda_r(o) \leqslant \lambda(u) \leqslant \lambda_a(o)$. By Constraint 3.3.1, $\lambda(u) = \mathsf{Roles}(u) = r'$. By Constraint 3.3.2, u can only activate r' in a session s, that is $\mathsf{Roles}(s) = r'$. By Constraint 3.3.12, $\lambda_r(o) = r_r$ and $\lambda_a(o) = r_a$. Since $r_r \leqslant r_a$, by Constraint 3.3.12, $\mathsf{Roles}(p) = [r_r, r_a]$. By Constraint 3.3.7,

$\mathsf{Roles}_E(p) = \mathsf{Roles}(p) = [r_r, r_a]$. Since $\lambda_r(o) \leqslant \lambda(u) \leqslant \lambda_a(o)$, $\mathsf{Roles}(s) \in [r_r, r_a]$. Hence $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) \neq \emptyset$. Therefore u is authorized for p in Σ'. ■

3.3.7 The discretionary security property

An information flow policy authorizes access purely on the basis of the labelling of the subject and object. For example, a subject s is authorized to append to an object o only if $\lambda(s) \leqslant \lambda(o)$. However, in many situations, it will be undesirable for a user to append to a file with a higher security label. The protection matrix is usually used to augment the information flow policy when defining authorization requirements in computer systems. We might enforce the above requirement in BLP simply by ensuring that $\texttt{append} \notin M_{s,o}$.

Existing work on implementing BLP using RBAC ignores the discretionary property [29, 69, 72, 73, 81], meaning that the range of policies that can be implemented are somewhat limited. We now show that 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 $r' > r$. Constraint 3.3.6 insists 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. To support discretionary security we replace Constraints 3.3.5 and 3.3.6 with the following constraints.

Constraint 3.3.13 *If* $p = (o, \texttt{read})$ *and* $(p, r) \in PA$, *then either* $p \in P^+$ *or* $p \in P^0$ *and* $\mathsf{Roles}_E(p) \subseteq \uparrow r$.

Constraint 3.3.14 *If* $p = (o, \texttt{append})$ *and* $(p, r) \in PA$*, then either* $p \in P^-$ *or* $p \in P^0$ *and* $\mathsf{Roles}_E(p) \subseteq {\downarrow}r$*.*

There is one distinct limitation to this approach. Namely, we cannot define our policies at the user-level, only at the role-level. This means that it is not possible to achieve the granularity of policy specification that is available in the BLP model via the protection matrix.[4] This is an inherent limitation of the RBAC paradigm, not the OP-RBAC model. One advantage of this approach over BLP is that policy specification is much more economical. This, of course, is one of the advantages of the RBAC paradigm. It is instructive to note that OP-RBAC can support this level of discretionary policy specification, while standard RBAC cannot.

3.3.8 Discussion

There have been several attempts to implement BLP models using role-based models [29, 69, 72, 73, 81]. Osborn *et al*'s approach [69, 72, 73] shows how the role-graph model can be configured to enforce information flow policies.

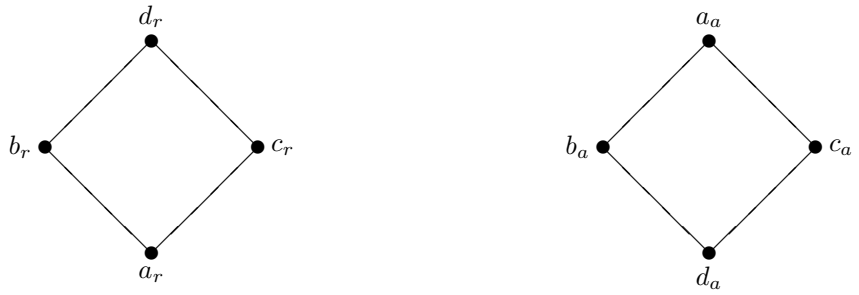
In their approach the lattice of security labels is defined separately and independently from the role graph. Each subject and object is assigned a security label, as in BLP. Then the *r-level* of a role r, denoted by r-level(r), is defined to be the least upper bound of the security labels of the objects 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.

[4] In fact our approach is equivalent to a BLP model in which the protection matrix has rows indexed by security levels and columns indexed by objects.

It has been noted that this approach creates a permission-permission conflict [88]. For example, a role r contains permissions to objects labeled with security labels (rts, as), where rts denotes "read top secret" and as denotes "append secret"; that is, r-level(r) is rts and a-level(r) is as. Therefore, the role r cannot be assigned to any user without violating information flow policies. In addition, 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* [73, 81]. They introduced two partial orderings, $\leqslant_r$ and $\leqslant_a$, on the set of roles. This gives rise to two hierarchies RH_r and RH_a, one for read roles and one for append roles. The append hierarchy RH_a is the dual of the read hierarchy RH_r: that is, $x \leqslant y$ in the append hierarchy if and only if $x \geqslant y$ in the read hierarchy. Figure 3.2 shows a simple example of the read hierarchy and the corresponding append hierarchy. We write x_r for role x in the read hierarchy and x_a for the corresponding role in the append hierarchy. Each pair of permissions $(o, \texttt{read})$ 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.

A number of constraints, similar to a number of ours were also defined. Note that, in BLP, every subject has a unique security label, and the concept of subjects corresponds to sessions in RBAC96 [83]. Hence, they require that each session has exactly two matching roles y_r and y_a. 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. Since RH_a is the dual of RH_r, the condition of granting an append permission is actually $y_a \leqslant x_a$ in original lattice



(a) RH_r (b) RH_a

Figure 3.2: A read hierarchy and the corresponding append hierarchy

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. Finally, the simulation of BLP_3 is not covered in their approach.

Crampton recently introduced the OP-RBAC model, and illustrated how OP-RBAC can be used to implement multi-level secure systems with the addition of few constraints to the basic model. The constrained OP-RBAC model simulates a more "general" version of BLP, where the read permission $p = (o, \texttt{read})$ and the append permission $p' = (o, \texttt{append})$ are assigned to the unique roles r_r and r_a respectively, and the security label of an object $\lambda(o)$ is defined to be a range $[r_r, r_a]$. Clearly, there are a number of possibilities for the security label of an object: if $r_r = r_a$, $\lambda(o) = r_r = r_a$; if $r_r < r_a$, $\lambda(o) = [r_r, r_a]$; if $r_a < r_r$, $\lambda(o) = \emptyset$. However, in the standard BLP models, the security label of an object is unique, which corresponds to the situation where $r_r = r_a$. On the other hand, it is rare to see any BLP models with the security label of an object being either $\emptyset$ or $[r_r, r_a]$, which makes it difficult to compare the security label of an object with the security label of a subject. In other words, the constrained OP-RBAC model only implements a simple version of BLP, that is, BLP_0.

We re-examined OP-RBAC, with the addition and modification of a few constraints, to provide greater correlation between role-based and mandatory access control. Com-

pared with previous attempts, our approach provides a more direct implementation of a number of different BLP models. 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. In addition, our approach provides a natural implementation of BLP_3 in terms of untrusted subjects, which has not been studied in most existing work. 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. Furthermore, we introduce BLP_4, an additional BLP model, that may provide some useful features unavailable in other BLP models. We have shown that BLP_4 can also be implemented in OP-RBAC.

3.4 Other applications

In this section, we 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 distinguish role activation and permission inheritance in a single OP-RBAC hierarchy.

3.4.1 Separation of duty

Separation of duty is widely considered to be a fundamental principle in computer security [25, 77]. In its simplest form, the principle requires that if a sensitive task comprises two steps, then the same user can not perform both steps. A typical example of separation of duty for a purchase order application is the requirement that creating (p) and approving (q) a purchase order can not be performed by the same user.

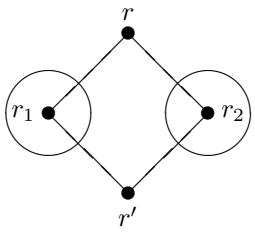
Separation of duty has always been an important consideration in role-based access control models. An approach commonly advocated in standard RBAC is to define

constraints on the authorizations of users to roles to ensure that no users can invoke both permissions p and q. This may be done through appropriate configuration of the UA relation (so called static separation of duty), or by limiting the roles that can be activated by users in a single session (dynamic separation of duty). Dynamic separation of duty is claimed to provide greater operational flexibility in practice [2]. Assume that p and q are assigned to roles r_1 and r_2 respectively, dynamic separation of duty requires that no user is permitted to activate both r_1 and r_2 in the course of any session, although it is possible for a user to be assigned to both roles. However, as we noted in Chapter 2, the role hierarchy may be incompatible with the enforcement of dynamic separation of duty constraints. That is, it is not possible to have a role r that is more senior than two roles r_1 and r_2 which are in dynamic separation of duty [62]. More specifically, assume that there exists some users who are able to activate roles r_1 and r_2 in different sessions, there is no means to assign these users to a common senior role r because activation of role r violates dynamic separation of duty with respect to r_1 and r_2. Therefore, these common users have to be explicitly assigned to roles r_1 and r_2, which goes against the idea of a role hierarchy to reduce administrative complexity.

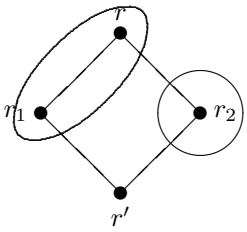
OP-RBAC introduced three different types of permissions and proposed a new mechanism for permission inheritance within a role hierarchy. This new approach of permission inheritance in OP-RBAC makes it possible to implement dynamic separation of duty constraints on two roles that have a common senior role and for a user to be assigned to or activate the senior role. The basic idea is to define sensitive permissions to be different types of permissions so that these permissions can not be aggregated to a single role. Formally, we require that $\mathsf{Roles}_E(p) \cap \mathsf{Roles}_E(q) = \emptyset$, which means no single role is authorized for both permissions p and q. Note that this condition does not exclude the case that there exists a role that is more senior than two roles to which sensitive permissions p and q are explicitly assigned, and this senior role can

be activated by a user without violating dynamic separation of duty constraints on its junior roles.

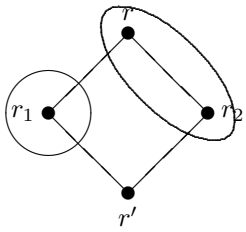
Figure 3.3 illustrates the seven ways of ensuring that for mutually exclusive permissions p and q assigned to roles r_1 and r_2 respectively, $\mathsf{Roles}_E(p) \cap \mathsf{Roles}_E(q) = \emptyset$. (The roles enclosed by a curve illustrate the effective set of roles for each permission.) The most obvious solution is to make p and q neutral permissions and assign them to roles r_1 and r_2 respectively, as shown in Figure 3.3(a). Therefore, u can be assigned to the more senior role r and activate r without acquiring the mutually exclusive permissions p and q. In addition, Figures 3.3(b)–3.3(g) 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.



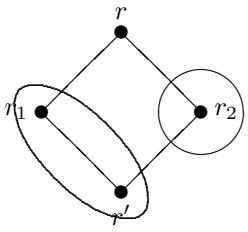
(a) $p, q \in P^0$



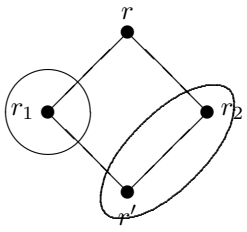
(b) $p \in P^+, q \in P^0$



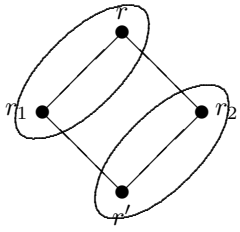
(c) $p \in P^0, q \in P^+$



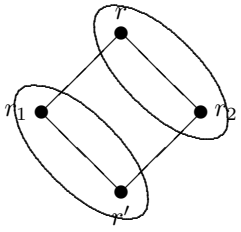
(d) $p \in P^-, q \in P^0$



(e) $p \in P^0, q \in P^-$



(f) $p \in P^+, q \in P^-$



(g) $p \in P^-, q \in P^+$

Figure 3.3: Implementing separation of duty using different types of permissions

Of course, we still need to define dynamic separation of duty constraints in OP-

RBAC to prevent, for example, u from activating roles r_1 and r_2 in the same session.[5] Formally, we require that for all $u \in U$, either $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(p) = \emptyset$ or $\mathsf{Roles}(s) \cap \mathsf{Roles}_E(q) = \emptyset$, where $u = \mathsf{User}(s)$. This conditions means that no user is allowed to activate a set of roles to which mutually exclusive permissions are assigned in any session.

3.4.2 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. That is, if $r \geqslant r'$ and $(p, r') \in PA$, then r is authorized for p.

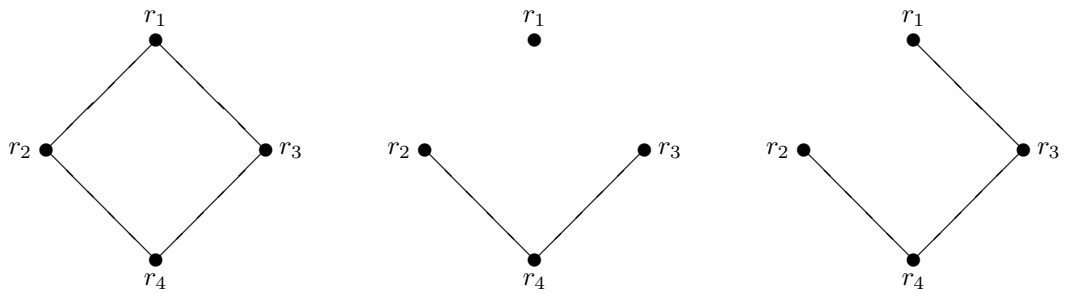
- A user assigned to a particular role can also activate any subordinate roles in the hierarchy; this is called the (*role*) *activation* aspect of role hierarchy. That is, if $r \geqslant r'$ and $(u, r) \in UA$, then u may activate (is authorized for) r'.

It has been observed that there are a number of situations where it is necessary to distinguish between role activation and permission usage [82]. Sandhu introduced an extended RBAC96 model (ERBAC96) [82] that defines a separate role activation hierarchy (denoted RH_a), a relation which is a superset of the permission usage hierarchy (denoted RH_u). Formally, these hierarchies are modelled as two partial orderings, $\leqslant_a$ and $\leqslant_u$, on the set of roles, where $r \leqslant_u r'$ implies that $r \leqslant_a r'$. In other words, if a user u assigned to role r' can use a permission by virtue of inheritance from role r, then u can also activate role r. Given $R' \subseteq R$, we define $\downarrow_a R' = \{r \in R : \exists r' \in R', r \leqslant_a r'\}$ and we define $\downarrow_u$, $\uparrow_a$ and $\uparrow_u$ in an analogous fashion. A user's interaction with the system is mod-

[5] We assume that the user u does not terminate a session, and log in with the other role. Auditing or some other mechanism is required to ensure that this loophole is not exploited when dynamic separation of duty constraints are used.

elled by a session s, where a user u activates a set of roles $\mathsf{Roles}(s) \subseteq \downarrow_a \mathsf{Roles}(u)$. The set of permissions for which u is authorized in a session s is defined to be $\mathsf{Prms}(\downarrow_u \mathsf{Roles}(s))$.

A typical application of the ERBAC96 model is to implement dynamic separation of duties between two roles that have a common senior role in an activation hierarchy. This has the same effect as OP-RBAC (described in the previous section) for solving the incompatibility issue that arises when there is a role hierarchy and dynamic separation of duty constraints. Consider the activation and usage hierarchies of ERBAC96 shown in Figures 3.4(a) and 3.4(c), respectively. Let us suppose that two roles r_2 and r_3 are mutually exclusive roles in a dynamic separation of duty constraint. We observe that a user assigned to role r_1, say, can activate r_1 in the activation hierarchy but does not inherit the permissions of r_2, because there is no inheritance relation between r_1 and r_2 in the usage hierarchy. Hence, a user does not obtain any mutually exclusive permissions that have been assigned to r_2 and r_3.



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

Figure 3.4: ERBAC96 activation and usage hierarchies

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 each permission is determined by the permission usage hierarchy, and the new permission assignment relation is determined by the usage hierarchy and the permission type. In particular, we can transform an ER-

BAC96 system (UA, PA, RH_u, RH_a, P) into a OP-RBAC system (UA, PA', RH_a, P') using the procedure in Figure 3.5.

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 add p to P^0 and (p, r) to PA^0, and for all $r' \in \uparrow_u r$, we add (p, r') to 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 add p to P^+ and (p, r) to PA^+;
5. Define $P' = P^+ \cup P^0$ and $PA' = PA^+ \cup PA^0$;
6. For all $p \in P'$ such that $(p, r) \in PA^0$ and $(p, r') \in PA^+$, we remove p from P^+, and for all $r'' \in \uparrow_u r'$, we add (p, r'') to PA^0, and remove (p, r') from PA^+.

Figure 3.5: Transforming ERBAC96 into OP-RBAC

We now show how the transformation works by taking the example of the ERBAC96 system illustrated in Figure 3.4 and the permission-role assignment relation in Figure 3.6(a). Firstly, we consider the example of permission usage hierarchy in Figure 3.4(b). Let us assume that the first role examined by the transformation procedure described above is role r_4. The first stage (Step 3) 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 permission 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 repeat the computations for roles r_2, r_3 and 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 3.6(b), where the third column contains two symbols: + and 0 that denotes the corresponding permissions are up permissions and neutral permissions respectively.

For the second usage hierarchy in Figure 3.4(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 6). Hence, we add (p_2, r_3) and (p_2, r_1) to PA^0 and delete (p_2, r_3) from PA^+ (Step 6). Finally, the new permission role assignment relation PA' is generated as shown in Figure 3.6(c).

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. 3.4(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. 3.4(c)

Figure 3.6: Transforming the ERBAC96 permission set and PA relation

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 ERBAC96 system for all possible access requests.

Theorem 3.4.1 *Let $\Sigma = (UA, PA, RH_a, RH_u, P)$ define an ERBAC96 system and $\Sigma' = (UA, PA', RH_a, P')$ define an OP-RBAC system derived from the ERBAC96 system in the manner described in Figure 3.5. Then for all $p \in P$, $\uparrow_u \mathsf{Roles}(p)$ in Σ is equal to $\mathsf{Roles}_E(p)$ in Σ'.*

Proof We first prove that $\uparrow_u \mathsf{Roles}(p) \subseteq \mathsf{Roles}_E(p)$. Let $r \in \uparrow_u \mathsf{Roles}(p)$: 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 $r \in \mathsf{Roles}_E(p)$. 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, since $RH' = RH_a$, $r \in \mathsf{Roles}_E(p)$. (Note that if $(p, r'') \in PA^0$ and $(p, r') \in PA^+$, then we

add (p, r) to PA^0, using Step 6, and hence $r \in \mathsf{Roles}_E(p)$.) Therefore, $\uparrow_u \mathsf{Roles}(p) \subseteq \mathsf{Roles}_E(p)$.

We now prove that $\mathsf{Roles}_E(p) \subseteq \uparrow_u \mathsf{Roles}(p)$. Let $r \in \mathsf{Roles}_E(p)$: 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 6). By definition, $r \in \uparrow_u \mathsf{Roles}(p)$. 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, $r \in \uparrow_u \mathsf{Roles}(p)$. Therefore $\mathsf{Roles}_E(p) \subseteq \uparrow_u \mathsf{Roles}(p)$. The result now follows.■

Corollary 3.4.1 *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 by Σ, if and only if there exists $r \in \mathsf{Roles}(s)$ such that $r \in \uparrow_u \mathsf{Roles}(p)$. By Theorem 3.4.1, $r \in \uparrow_u \mathsf{Roles}(p)$ if and only if $r \in \mathsf{Roles}_E(p)$. ■

In summary, permission usage requirements in the ERBAC96 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 adds somewhat 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 have shown how an ERBAC96 system can be transformed into an equivalent (in terms of what requests are authorized) OP-RBAC system.

3.5 Conclusion

We have considered three 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 a number of different BLP models using role-based techniques. Our approach provides the first such implementation that supports the assignment of compound permissions (both read and append access to objects). Equally importantly, we demonstrate how it is possible to incorporate limited support for the discretionary security property of BLP, something that no existing work is able to do. In addition we described a new version of BLP (that can also be implemented using OP-RBAC), where each object associates with a *read* security label and an *append* security label. This provides support for some useful separation of duty properties at the object, where no role can both read and append a file simultaneously.

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 is able to do this with a single role hierarchy.

Finally, we have described a way of supporting the separation between permission usage and role activation within a single hierarchy. We have defined a transformation that generates a OP-RBAC system that is equivalent to a given ERBAC96 system, in the sense that the same set of requests is authorized. The success of this transformation is based on the constraint ERBAC96 imposed on the use of role hierarchies, that is, the usage hierarchy is restricted to be a subset of the activation hierarchy. In contrast, OP-RBAC can not support any arbitrary configurations of activation and usage hierarchies which is permitted in the GTRBAC model, for example.

On the other hand, ERBAC96 and GTRBAC certainly have merit and deal satisfactorily with both selective inheritance and dynamic separation of duty constraint

as we noted in this chapter. However, the motivation for the specific ways in which these hierarchies are defined is unclear. In addition, GTRBAC introduced temporal constraints to various components of RBAC, including role hierarchies. The syntax for the model is rather complicated and the semantics defining the interaction between temporal constraints on the *UA*, *PA*, roles, and role hierarchies are not clearly defined. In the next chapter, we will consider these issues in more detail and present some solutions.

Chapter 4

Spatio-Temporal RBAC

Role-based access control has attracted considerable research interest in recent years due to its potential ability to model organizational structure and to reduce administrative overheads. The most distinctive and important feature of the RBAC approach is the role hierarchy, which generally supports two different types of inheritance: role activation and permission usage. For example, in the RBAC96 model, if a user u is assigned to role r (that is, $(u, r) \in UA$), then u is implicitly assigned to all roles in the set $\downarrow r = \{r' \in R : r' \leqslant r\}$, and u can create a session by activating the roles in any subset of $\downarrow r$. Similarly, if a permission p is assigned to role r (that is, $(p, r) \in PA$), then p is implicitly assigned to all roles in the set $\uparrow r = \{r' \in R : r' \geqslant r\}$. Essentially, the role hierarchy allows the users who activate senior roles to inherit all permissions of their junior roles, and conversely ensures that the users of junior roles inherit any prohibitions (constraints) that apply to their senior roles. We believe it is important to preserve this inheritance semantics when considering extensions to role-based access control.

There has been some work on extending role-based access control to include multiple role hierarchies [58, 67, 82]. Sandhu introduced an extended RBAC96 model

(ERBAC96) [82], in which two different orderings were defined on the set of roles, one governing role activation and one governing permission usage. In addition, Joshi *et al* introduced a generalized temporal RBAC (GTRBAC) model [58] which also distinguishes between permission usage and role activation by defining three different orderings on the set of roles. However, we believe the way of defining those role hierarchies in both models somewhat ignored the inheritance semantics on the combination of these hierarchies.

In this chapter, we consider an alternative way of distinguish usage and activation hierarchies by developing a novel extended RBAC model (ERBAC07). In particular, the authorization semantics of ERBAC07 is based on our graph-based interpretation of RBAC_1 that is a simple way to see how access requests are evaluated in RBAC96. Compared with ERBAC96 and GTRBAC, ERBAC07 has intuitive inheritance semantics and clear authorization semantics.

The GTRBAC model caters for time-dependent access control policies. More generally, the emergence of mobile and ubiquitous computing environments poses new demands on access control mechanisms, because the decision to grant access may depend on contextual information, such as the location of the user and the time at which access requests are made. It may be appropriate, for example, to limit the time and places at which a particular role can be activated.

Several context-based RBAC models have been defined in recent years [13, 14, 28, 43, 49, 58, 76, 86]. Each of these models introduces extensions to the basic role-based model in which components may be associated with general contextual constraints [28, 43, 86], temporal constraints [13, 58], spatial constraints [14, 49], or spatio-temporal constraints [76]. However, none of these models accurately captures the interaction between spatio-temporal constraints and inheritance in the RBAC model: indeed, all of them have one or more of the following limitations.

- No existing model has clear semantics for inheritance in the role hierarchy in the presence of spatio-temporal constraints. This means that there is no way of designing an algorithm for deciding access requests.

- Existing models are extremely complicated. GTRBAC [58] and the spatio-temporal RBAC model of Ray and Toahchoodee [76] define a large number of predicates to specify temporal constraints and spatio-temporal constraints, respectively. The relationship between these predicates is often unclear, again making it difficult to see how access requests should be evaluated in such models.

- Conflicts and ambiguity may occur in existing models. Conflicts may arise among the constraints defined in spatio-temporal RBAC [76], for example.

- Existing models lack compatibility with RBAC96 and the closely related ANSI-RBAC standard. It is not at all clear how to translate the predicates used in GTRBAC [58] and spatio-temporal RBAC [76], for example, to the entities and relations used in the ANSI-RBAC standard and RBAC96.

In summary, we would argue that existing models focus far too much on syntax, and far too little on semantics.

In this chapter, we explore a number of ways to define spatio-temporal constraints on the RBAC models. The majority of the material in this chapter has already been published [21, 22]. We firstly develop three spatio-temporal RBAC models by extending the basic RBAC_1 model with very little additional syntax. The authorization semantics of these three models are based on the graph-based formalism of RBAC96, and are varied in the extent to which RBAC entities and relations are constrained by spatio-temporal restrictions. We also extend these models to include spatio-temporal requirements for ERBAC07. Compared with GTRBAC and the spatio-temporal RBAC model by Ray and Toahchoodee, these simple, expressive, flexible spatio-temporal RBAC models have

clear, well-defined semantics and are designed to be compatible with RBAC96 and the ANSI-RBAC standard.

Unlike existing work, we consider the practical implications of using our spatio-temporal models. The introduction of spatio-temporal constraints increases the complexity of evaluating requests, particularly in the presence of a role hierarchy and enabling conditions on roles. We suggest pre-computing transitive closure of (part of) the RBAC graph to improve requests' response time. On the other hand, the interaction between spatio-temporal constraints and inheritance is complex, which makes it difficult to specify spatio-temporal constraints in the presence of a role hierarchy. We propose two strategies to mitigate these difficulties: the use of flat spatio-temporal model and the omission of constraints on roles and the role hierarchy.

The rest of this chapter is organized as follows. In the next section, we introduce a novel graph-based formalism to define the semantics of RBAC_1. In Section 4.2, we define the ERBAC07 model with RBAC_1-style syntax and graph-based semantics. In Section 4.3, we formally define the $\text{RBAC}^{=}_{ST}$, RBAC^{+}_{ST} and RBAC^{-}_{ST} models, and introduce the notion of trusted entities. We also demonstrate the use of RBAC_1-style syntax to encode spatio-temporal RBAC models, and illustrate how to integrate our spatio-temporal functions into the ANSI-RBAC standard. In Section 4.4, we introduce the $\text{ERBAC}^{=}_{ST}$, ERBAC^{+}_{ST} and ERBAC^{-}_{ST} models for ERBAC07. In Section 4.5, we consider practical implementation of our spatio-temporal RBAC and ERBAC models, focusing on mitigating difficulties arising from different interactions between spatio-temporal constraints and a role hierarchy. In Section 4.6, we discuss possible representations of spatial and temporal domains, and give concrete examples of spatial $\text{RBAC}^{=}_{ST}$, temporal $\text{RBAC}^{=}_{ST}$ and spatio-temporal $\text{ERBAC}^{=}_{ST}$. Section 4.7 compares our work with related work in the literature. Finally, we summarize the work of the chapter in Section 4.8.

4.1 Graph-based formulation of RBAC_1

Recall that the RBAC_1 model defines a set of roles R, a role hierarchy $RH \subseteq R \times R$, a user-role assignment relation $UA \subseteq U \times R$ (where U is a set of users), and a permission-role assignment relation $PA \subseteq P \times R$ (where P is a set of permissions). We write $\leqslant$ to denote the transitive reflexive closure of the RH relation; $(R, \leqslant)$ is a partially ordered set (since the directed graph of the role hierarchy relation is assumed to be acyclic). We represent RBAC_1 state as a tuple (UA, PA, RH). We now introduce a novel graph-based formulation of RBAC_1, which we believe to be simple and intuitive specification of the basic components of the RBAC_1 model. As we will see, this formulation can be readily extended to include spatio-temporal restrictions.

We construct an acyclic, directed graph $G = (V, E)$, where $V = U \cup R \cup P$, and $E = UA \cup PA \cup RH$. In other words, each vertex v represents an entity, such as a user u, a role r or a permission p in a RBAC_1 system, and each directed edge $e = (v_i, v_j)$ represents a relationship between two entities v_i and v_j; specifically, $(v_i, v_j) \in E$ if and only if (precisely) one of the following conditions holds

$$(v_i, v_j) \in UA,$$

$$(v_j, v_i) \in RH,$$

$$(v_j, v_i) \in PA.$$

An *authorization path* (or *au-path*) between v_1 and v_n is a sequence of vertices $v_1, \ldots, v_n$ such that $(v_i, v_{i+1}) \in E$, $i = 1, \ldots, n-1$. Hence, a user u can activate a role r if there is an au-path between u and r; a role r is authorized for permission p if there is an au-path between r and p; and a user u is authorized for permission p if there is an au-path between u and p. To summarize, we introduce the following definition.

Definition 4.1.1 *An entity $v \in U \cup R$ is* RBAC_1-authorized *for $v' \in R \cup P$ if and only if there exists an au-path $v = v_1, v_2, \ldots, v_n = v'$.*[1]

We also explicitly define the RBAC_1 (authorization) semantics in terms of the graph formulation.

Definition 4.1.2 *The* semantics *of a $RBAC_1$ state (UA, PA, RH), denoted $Auth[UA, PA, RH]$, is the set of requests that is authorized by this state. That is, $Auth[UA, PA, RH] = \{(u, p) \in U \times P$: there exists an au-path $u, \ldots, p\}$.*

4.2 ERBAC07

In this section, we construct the ERBAC07 model, extending the syntax used in RBAC_1. We define the semantics of ERBAC07 by extending the graph-based formulation of RBAC_1. In doing so, the ERBAC07 model has a clear and well-defined semantics, in contrast to many existing RBAC extensions, and provides a context for studying spatio-temporal requirements in Section 4.4.

4.2.1 Syntax

The ERBAC07 model defines a set of roles R, a user-role assignment relation $UA \subseteq U \times R$, and a permission-role assignment relation $PA \subseteq P \times R$. We replace the standard role hierarchy relation in the RBAC_1 model with a new relation $RH \subseteq R \times R \times \{a, u\}$. We define $RH_a = \{(r, r') : (r, r', a) \in RH\}$ and $RH_u = \{(r, r') : (r, r', u) \in RH\}$. We call RH_a the (*role*) *activation hierarchy* and RH_u the (*permission*) *usage hierarchy*. We write $\leqslant_a$ to denote the reflexive, transitive closure of RH_a and $\leqslant_u$ to denote the reflexive, transitive closure of RH_u. In other words, $\leqslant_a$ and $\leqslant_u$ are modeled

[1] Note that $r \in R$ is RBAC_1-authorized for $r' \in R$ means that $r \geqslant r'$ in the role hierarchy.

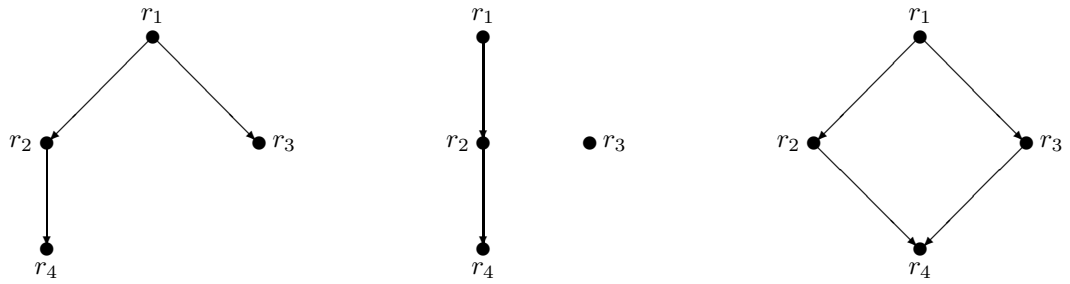
as two partial orderings on the set of roles R. We represent ERBAC07 state as a tuple (UA, PA, RH_a, RH_u).

ERBAC96 [82] introduces the distinction between usage and activation hierarchies and restricts the form that the activation hierarchy can take: $r \leqslant_u r'$ implies that $r \leqslant_a r'$. The effect of this restriction is to guarantee that if r' inherits permissions assigned to role r by virtue of the fact that $r \leqslant_u r'$, then any user that can activate r' should also be able to activate role r. However, no particular motivation for this restriction is provided by Sandhu.

We now consider whether any restriction is necessary and, if so, what form such a restriction should take. Consider the following statement: if a user can activate role r and $r' \leqslant_a r$ then the permissions for which r' is authorized should not be a superset of the permissions for which r is authorized. This statement is certainly reasonable at an intuitive level and an immediate consequence of this statement is that if $r' \leqslant_a r$ then we require $r \not\leqslant_u r'$ (otherwise r' is implicitly authorized for all the permissions for which r is explicitly and implicitly authorized). A logically equivalent statement is that $r \leqslant_u r'$ implies that $r' \not\leqslant_a r$. In other words, we require that the following constraint on $\leqslant_a$ and $\leqslant_u$ be satisfied.

Constraint 4.2.1 *If $r \leqslant_u r'$, then $r \parallel_a r'$ or $r \leqslant_a r'$, where $r \parallel_a r'$ denotes roles r and r' are incomparable in the activation hierarchy.*

Clearly, this constraint imposed on ERBAC07 offers more flexibility than ERBAC96. We will further compare it with ERBAC96 and GTRBAC in Section 4.2.3. We now use graph-based formalism to explain the authorization semantics of ERBAC07.



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

Figure 4.1: A graphical representation of ERBAC07 states

4.2.2 Semantics

We also use a graph which is a simple and intuitive way to represent features of ERBAC07. In order to satisfy Constraint 4.2.1, we construct an *acyclic*, directed graph $G = (V, E)$, where $V = U \cup R \cup P$, and $E = UA \cup PA \cup RH_a \cup RH_u$. An *activation path* (or *a-path*) between v_1 and v_n is defined to be a sequence of vertices $v_1, \ldots, v_n$ such that $(v_1, v_2) \in UA$ and $v_{i+1} \leqslant_a v_i$ for $i = 2, \ldots, n-1$. A *usage path* (or *u-path*) between v_1 and v_n is defined to a sequence of vertices $v_1, \ldots, v_n$ such that $v_{i+1} \leqslant_u v_i$ $(i = 1, \ldots, n-2)$ and $(v_n, v_{n-1}) \in PA$. In ERBAC07:

- $v \in U$ may activate role $v' \in R$ if and only if there exists an a-path $v = v_1, v_2, \ldots, v_n = v'$;
- $v \in R$ is authorized for permission $v' \in P$ if and only if there exists a u-path $v = v_1, v_2, \ldots, v_n = v'$;
- $v \in U$ is authorized for permission $v' \in P$ if and only if there exists a path $v = v_1, v_2, \ldots, v_i, \ldots, v_n = v'$ such that $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an a-path, and $v_i, \ldots, v_n$ is a u-path.

We say $v_1, \ldots, v_n$ is an au-path in ERBAC07 if $v_1, \ldots, v_n$ is either an a-path, or a u-path, or the concatenation of an a-path and a u-path. To summarize, we introduce

the following definition.

Definition 4.2.1 *An entity $v \in U \cup R$ is* ERBAC07-authorized *for $v' \in R \cup P$ if and only if there exists an au-path $v = v_1, \ldots, v_n = v'$.*

Definition 4.2.2 *The* semantics *of a ERBAC07 state (UA, PA, RH_a, RH_u), denoted $Auth[UA, PA, RH_a, RH_u]$, is the set of requests that is authorized by this state. That is, $Auth[UA, PA, RH_a, RH_u] = \{(u, p) \in U \times P$: there exists an au-path $u, \ldots, p\}$.*

A typical application of the ERBAC07 model (as for ERBAC96) is to facilitate the implementation of dynamic separation of duty constraints on roles that have a common senior role. Consider the activation hierarchy of ERBAC07 shown in Figure 4.1(a). Suppose that no user should have the permissions of both roles r_2 and r_3 available in the course of any session. (That is, the roles r_2 and r_3 are in dynamic separation of duty.) In order to achieve this we could define the usage hierarchy of ERBAC07 as shown in Figure 4.1(b), where a user assigned to the role r_1 can activate the role r_1 but does not inherit the permissions of r_3.

Li *et al* [64] recently discussed inheritance semantics of a role hierarchy for the ANSI RBAC standard, and suggested a useful application when only permission inheritance is used by an RBAC system. Unlike ERBAC96, ERBAC07 allows configurations of activation and usage hierarchies to achieve the same effect as a single role hierarchy with only permission inheritance semantics. Consider the configuration of activation and usage hierarchies shown in Figures 4.1(a) and 4.1(c). A user u who is assigned to role r_3 is authorized for permissions assigned to roles r_3 and r_4, but u is not authorized for (or allowed to activate) r_4. In other words, u can use permissions assigned to r_4 without knowing the existence of r_4. This is particularly useful for a system where the intricate details of how permissions are set up through roles need to be partially hidden from certain users [64].

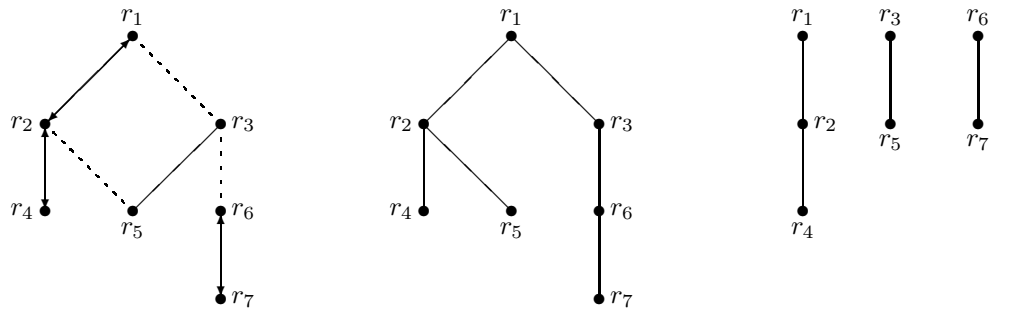
4.2.3 ERBAC96, GTRBAC and ERBAC07

In this section, we consider the two most significant existing approaches to multiple role hierarchies, the ERBAC96 model [82] and the GTRBAC model [58], and briefly compare them to our ERBAC07 model. Roughly speaking, we conclude that the ERBAC96 model is too restrictive and that the GTRBAC model is too general.

The ERBAC96 model is identical to the ERBAC07 model in terms of syntax, with the exception that ERBAC96 has the following constraint: $r \leqslant_u r'$ implies that $r \leqslant_a r'$. In other words, the ERBAC96 model introduces a separate activation hierarchy, a relation which is a superset of the usage hierarchy. The motivation for imposing such a constraint is to simulate mandatory access control systems using role-based access control, and to facilitate the implementation of dynamic separation of duty in hierarchical RBAC. Hence, we think the ERBAC96 model is rather limited and excludes some configurations of RH_a and RH_u that might offer useful applications. In contrast, we develop the ERBAC07 model by considering what are the appropriate ways to distinguish the activation and usage hierarchies, rather than as a mechanism for implementing special requirements. In other words, we define a weaker constraint on RH_a and RH_u, making ERBAC07 more general than ERBAC96, thereby ERBAC07 inherits all the applications of ERBAC96. Most importantly, the ERBAC07 model has graph-based semantics that can be naturally extended to include spatio-temporal requirements, unlike ERBAC96.[2]

The GTRBAC model defines a "hybrid" role hierarchy that contains three different types of role hierarchy relationships: activation hierarchy $\leqslant_a$, usage hierarchy $\leqslant_u$ and permission-activation hierarchy $\leqslant$. Unlike ERBAC07, GTRBAC does not impose any constraints on the activation and the usage hierarchies. In other words, GTRBAC allows to define some configurations of RH_a and RH_u such as $r \leqslant_a r'$ and $r' \leqslant_u r$.

[2]We certainly could define similar semantics for ERBAC96.



(a) A hybrid hierarchy (b) An activation hierarchy (c) A usage hierarchy

Figure 4.2: Decomposing a GTRBAC hierarchy into ERBAC07 hierarchies

These configurations mean that a user who activates a junior role is able to acquire more permissions than a senior role, which is counter-intuitive and incompatible with the inheritance semantics of RBAC96. However, the constraint we introduced in ERBAC07 prohibits the occurrence of these configurations of RH_a and RH_u.

In addition, the permission-activation hierarchy is redundant in GTRBAC and can be defined in terms of other two hierarchies, that is $x \leqslant y$ if and only if $x \leqslant_a y$ and $x \leqslant_u y$. Figure 4.2(a) shows a hybrid hierarchy used previously to illustrate the GTRBAC model [36]: a solid edge denotes an element of the usage hierarchy; a dashed edge indicates an element of the activation hierarchy; and a double-headed arrow indicates an element of the permission-activation hierarchy. It is obvious that the hybrid hierarchy can be simply expressed by the activation and usage hierarchies shown in Figures 4.2(b) and 4.2(c).

Furthermore, GTRBAC does not have explicit authorization semantics, which means it is not clear how access requests are answered in a GTRBAC system. In contrast, ERBAC07 has a simple method of distinguishing activation and usage hierarchies, and has clear authorization semantics.

4.3 Spatio-temporal RBAC

We assume the existence of the usual RBAC_1 sets and relations: U, R, P, UA, PA, and RH; we write V to denote $U \cup R \cup P$ and E to denote $UA \cup PA \cup RH$. We also assume the existence of a spatio-temporal domain $\mathcal{D}$: $d \in \mathcal{D}$ represents a point in space-time; $D \subseteq \mathcal{D}$ represents a collection of points in space-time.[3]

4.3.1 $\text{RBAC}^{=}_{ST}$: the standard model

The *standard spatio-temporal RBAC model* (or $\text{RBAC}^{=}_{ST}$) augments the RBAC_1 model with a function $\lambda : V \to 2^{\mathcal{D}}$. For $v \in V$, $\lambda(v) \subseteq \mathcal{D}$ denotes the set of points in space-time at which v is "enabled". In particular,

- if $u \in U$, then $\lambda(u)$ denotes the set of points in space-time at which u may create a session;
- if $r \in R$, then $\lambda(r)$ denotes the set of points in space-time at which r may be activated in a session;
- if $p \in P$, then $\lambda(p)$ denotes the set of points in space-time at which p may be granted to a user.

Given a path $v_1, \ldots, v_n$ in the labeled graph $G = (V, E, \lambda)$, we write $\widehat{\lambda}(v_1, \ldots, v_n) \subseteq \mathcal{D}$ to denote $\bigcap_{i=1}^{n} \lambda(v_i)$. In other words, $\widehat{\lambda}(v_1, \ldots, v_n)$ is the set of points at which every vertex v_i is enabled. When the context is clear, we will write $\widehat{\lambda}(v_1, v_n)$ for $\widehat{\lambda}(v_1, \ldots, v_n)$.

Definition 4.3.1 *An entity $v \in U \cup R$ is* $\text{RBAC}^{=}_{ST}$-authorized *for $v' \in R \cup P$ at point $d \in \mathcal{D}$ if and only if there exists an au-path $v = v_1, v_2, \ldots, v_n = v'$ and $d \in \widehat{\lambda}(v, v')$.*

Remark 4.3.1 *Recall that, $(v, v') \in E$ if and only if $(v, v') \in UA$ or $(v', v) \in RH$ or $(v', v) \in PA$. Consider a simple example: given $(u, r) \in UA$ and $\widehat{\lambda}(u, r) = \emptyset$, by*

[3] In Section 4.6 we elaborate on possible representations of $\mathcal{D}$. For the purposes of the discussion in this section, it is sufficient to assume the existence of some abstract spatio-temporal domain.

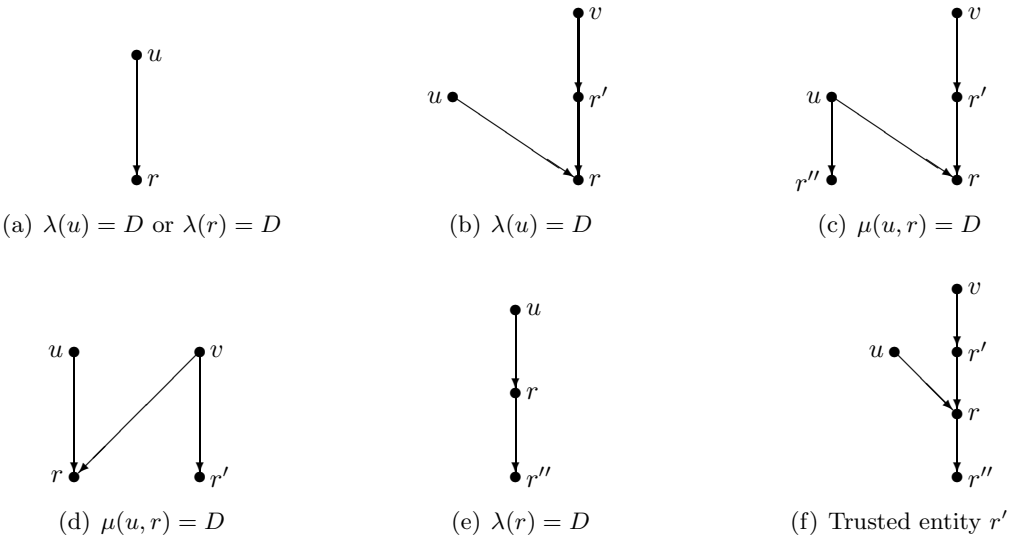
Definition 4.3.1, u is not $RBAC^{=}_{ST}$-authorized for r at any point $d \in \mathcal{D}$. In other words, u may not activate r if $\lambda(u, r) = \emptyset$, even if $(u, r) \in UA$. Hence, there is no way to authorize u for r if u and r do not have any common enabling points. In order to respond to this, more generally, if $e = (v, v') \in E$, then we assume that $\lambda(v) \cap \lambda(v') \neq \emptyset$, otherwise the edge is "wasted" as far as authorization is concerned.

We now introduce a running example, which will be used to motivate the additional models that we define. Let us assume that we want to express the following spatio-temporal constraints:

- any user assigned explicitly to role r can only activate this role in spatio-temporal domain $D \subseteq \mathcal{D}$;
- any user assigned explicitly to role r' can activate role r from any point $d \in \mathcal{D}$;
- any user may activate role r'' from any point $d \in \mathcal{D}$.

For concreteness, r might be a clerical role and users occupying this role may only activate this role if they are in some particular part of the office building. In contrast, r' is a managerial role and a user occupying this role may activate the clerical role when she is sitting in her own office (or anywhere else); r'' is a general employee role and can be activated from anywhere in the office.

Figure 4.3 illustrates six directed graphs for different user-role assignments and role hierarchies: vertices u and v represent users, and vertices r, r' and r'' represent roles. It is obvious that we can encode the above requirements using $\text{RBAC}^{=}_{ST}$ for the configurations shown in Figures 4.3(a) and 4.3(b). In particular, for Figure 4.3(a), we could define $\lambda(u) = D$ or $\lambda(r) = D$ or $\lambda(u) = \lambda(r) = D$. However, for Figure 4.3(b), we must define $\lambda(u) = D$, as user v, assigned to role r', is allowed to activate role r at any $d \in \mathcal{D}$.



(a) $\lambda(u) = D$ or $\lambda(r) = D$ (b) $\lambda(u) = D$ (c) $\mu(u, r) = D$

(d) $\mu(u, r) = D$ (e) $\lambda(r) = D$ (f) Trusted entity r'

Figure 4.3: RBAC_1 configurations and their effect on spatio-temporal configurations

Now consider the configuration in Figure 4.3(c), in which u is also assigned to role r''. Since u is allowed to activate r'' at any $d \in \mathcal{D}$, we can not define $\lambda(u) = D$; nor can we set $\lambda(r) = D$, as v is allowed to activate r at any $d \in \mathcal{D}$. Hence, we require a spatio-temporal constraint on edge (u, r). In other words, $\text{RBAC}^{=}_{ST}$ is not sufficiently expressive for certain RBAC_1 configurations and spatio-temporal requirements. For this reason, we now introduce a second model.

4.3.2 RBAC^{+}_{ST}: the strong model

The *strong spatio-temporal RBAC model* (RBAC^{+}_{ST}) augments the $\text{RBAC}^{=}_{ST}$ model with a function $\mu : E \to 2^{\mathcal{D}}$. For $e = (v, v') \in E$, $\mu(v, v')$ denotes the set of points in space-time at which the association between v and v' is enabled. In particular,

- if $(u, r) \in UA$, then $\mu(u, r)$ denotes the set of points in space-time at which u is assigned to r;
- if $(r, r') \in RH$, then $\mu(r', r)$ denotes the set of points in space-time at which r' is senior to r;

- if $(p, r) \in PA$, then $\mu(r, p)$ denotes the set of points in space-time at which p is assigned to r.

Given a path $v_1, \ldots, v_n$ in the labeled graph $G = (V, E, \lambda, \mu)$, we write $\widehat{\mu}(v_1, \ldots, v_n)$ to denote $\bigcap_{i=1}^{n-1} \mu(v_i, v_{i+1})$. Note that the semantics of $\text{RBAC}^{=}_{ST}$ imply that an edge can only be enabled if both end points are enabled. Similarly, in RBAC^{+}_{ST}, for $e = (v, v') \in E$, we require that $\mu(v, v') \subseteq \lambda(v) \cap \lambda(v')$. Hence, $\widehat{\mu}(v_1, \ldots, v_n)$ is the set of points at which every node and every edge in the path $v_1, \ldots, v_n$ is enabled. When the context is clear, we will write $\widehat{\mu}(v_1, v_n)$ for $\widehat{\mu}(v_1, \ldots, v_n)$.

Definition 4.3.2 *An entity $v \in U \cup R$ is* RBAC^{+}_{ST}-authorized *for $v' \in R \cup P$ at point $d \in \mathcal{D}$ if and only if there exists an au-path $v = v_1, v_2, \ldots, v_n = v'$ and $d \in \widehat{\mu}(v, v')$.*

Remark 4.3.2 *As in $\text{RBAC}^{=}_{ST}$, if there exists an edge $e = (v, v') \in E$ such that $\mu(v, v') = \emptyset$, by Definition 4.3.2, v is not RBAC^{+}_{ST}-authorized for v'. Therefore, if $e = (v, v') \in E$, then we assume that $\emptyset \subset \mu(v, v') \subseteq \lambda(v) \cap \lambda(v')$.*

Note that $\text{RBAC}^{=}_{ST}$ is a special case of RBAC^{+}_{ST} in which $\mu(v, v')$ is defined to be $\lambda(v) \cap \lambda(v')$. In other words, any spatio-temporal constraints we can encode in $\text{RBAC}^{=}_{ST}$, we can also encode in RBAC^{+}_{ST}.

Consider Figures 4.3(c) and 4.3(d), we can define $\mu(u, r) = D$ to express our spatio-temporal requirements in RBAC^{+}_{ST}. However, neither RBAC^{+}_{ST} nor $\text{RBAC}^{=}_{ST}$ can be used to express these requirements given the configuration in Figure 4.3(e). In particular, we cannot define $\lambda(r) = D$ in $\text{RBAC}^{=}_{ST}$ or $\mu(u, r) = D$ in RBAC^{+}_{ST} because this will only allow u to activate r'' at points $d \in D$ rather than the requirement $d \in \mathcal{D}$. We now introduce a third model with weaker restrictions on valid authorization paths.

4.3.3 RBAC^{-}_{ST}: the weak model

Like $\text{RBAC}^{=}_{ST}$, the *weak spatio-temporal RBAC model* (or RBAC^{-}_{ST}) augments the RBAC_1 model with a function $\lambda : V \to 2^{\mathcal{D}}$. In RBAC^{-}_{ST}, the authorization semantics are different from those in $\text{RBAC}^{=}_{ST}$.

Definition 4.3.3 *In $RBAC^{-}_{ST}$:*

- *a user $v \in U$ is $RBAC^{-}_{ST}$-authorized for role $v' \in R$ at point $d \in \mathcal{D}$ if and only if there exists an au-path $v = v_1, v_2, \ldots, v_n = v'$, and $d \in \lambda(v) \cap \lambda(v')$;*
- *a role $v \in R$ is $RBAC^{-}_{ST}$-authorized for permission $v' \in P$ at point $d \in \mathcal{D}$ if and only if there exists an au-path $v = v_1, v_2, \ldots, v_n = v'$, and $d \in \lambda(v) \cap \lambda(v')$;*
- *a user $v \in U$ is $RBAC^{-}_{ST}$-authorized for permission $v' \in P$ at point $d \in \mathcal{D}$ if and only if there exists an au-path $v = v_1, v_2, \ldots, v_i, \ldots, v_n = v'$ such that $v_i \in R$ for some i, and $d \in \lambda(v) \cap \lambda(v_i) \cap \lambda(v')$.*

In other words, an entity $v \in U \cup R$ is RBAC^{-}_{ST}-authorized for another entity $v' \in R \cup P$ if v is RBAC_1-authorized for v', and both entities v and v' are enabled.[4] There is no requirement that all intermediate nodes on the path are enabled. These semantics appear to be closest to those defined in GTRBAC and the model of Ray and Toahchoodee. However, we would argue that RBAC^{-}_{ST} has the least intuitive semantics: why is it appropriate to ignore the enabling conditions on intermediate roles? There may be occasions when it is convenient to do so, as in Figure 4.3(e), but ignoring the intermediate roles is unlikely to be appropriate in many situations, and is inconsistent with the usual interpretation of inheritance in a role hierarchy. We would argue that the standard or strong models, in which enabling conditions are inherited up the hierarchy, are more closely aligned with standard RBAC semantics.

[4] In the case where $v \in U$ and $v' \in P$, we require not only both entities v and v' are enabled, but also there exists an enabled role in the au-path from v to v'.

Let $G = (V, E, \lambda, \mu)$ be a graph for an RBAC^{+}_{ST} configuration. If a request (u, p) is RBAC^{+}_{ST}-authorized, then (u, p) is $\text{RBAC}^{=}_{ST}$-authorized. And if a request (u, p) is $\text{RBAC}^{=}_{ST}$-authorized, then (u, p) is RBAC^{-}_{ST}-authorized. The former condition is satisfied because of $\widehat{\mu}(u, p) \subseteq \widehat{\lambda}(u, p)$, and the latter $\widehat{\lambda}(u, p) \subseteq \lambda(u) \cap \lambda(r) \cap \lambda(p)$ for some role $r \in R$. In other words, given a graph $G = (V, E, \lambda, \mu)$, the authorization semantics of RBAC^{-}_{ST} (in terms of what requests are authorized) is least restrictive, and those of RBAC^{+}_{ST} is most restrictive.

Consider Figure 4.3(e). Using the weak model, we can define $\lambda(r) = D$ to realize our spatio-temporal requirements. However, we cannot express our spatio-temporal requirements for the configuration shown in Figure 4.3(f) using any of the models we have defined so far. If we use RBAC^{-}_{ST}, then we require that $\lambda(r) = D$ in order to restrict u's activation of r. This, in turn means that v is unable to activate r from any point $d \notin D$. However, if we use $\text{RBAC}^{=}_{ST}$ or RBAC^{+}_{ST}, we must define $\lambda(u) = D$ or $\mu(u, r) = D$, which means that u is unable to activate r'' from any point $d \notin D$. Hence we introduce the notion of *trusted entities*.

4.3.4 Trusted entities

A trusted entity may be a user or a role; we write $T \subseteq U \cup R$ to denote the set of trusted entities. For an entity $t \in T$, the enabling constraints on nodes/edges in the authorization path from t are ignored.[5] Trusted entities may be introduced to the standard, strong or weak model.

Definition 4.3.4 *An entity $v \in U \cup R$ is* $\text{RBAC}^{=}_{ST}$-authorized *for $v' \in R \cup P$ at point $d \in \mathcal{D}$ if and only if there exists an au-path $v = v_1, \ldots, v_j, \ldots, v_n = v'$ such that $v_j \in T$*

[5]The interpretation of a trusted entity is similar to that of a privileged method in the Java runtime environment (JRE). The stackwalking algorithm, which is used to perform access control in the JRE, normally examines the permissions of every method on the stack. Access is only granted if every method on the stack has the requested permission. However, the stackwalk terminates prematurely if a privileged method is encountered on the stack (thereby ignoring any methods lower down the stack that may not have the requested permission).

and $d \in \widehat{\lambda}(v, v_j)$*, or there exists an au-path* $v = v_1, v_2, \ldots, v_n = v'$ *and* $d \in \widehat{\lambda}(v, v')$.

Definition 4.3.5 *An entity* $v \in U \cup R$ *is* RBAC^{+}_{ST}-authorized *for* $v' \in R \cup P$ *at point* d *if and only if there exists an au-path* $v = v_1, \ldots, v_j, \ldots, v_n = v'$ *such that* $v_j \in T$ *and* $d \in \widehat{\mu}(v, v_j)$*, or there exists an au−path* $v = v_1, v_2, \ldots, v_n = v'$ *and* $d \in \widehat{\mu}(v, v')$.

Definition 4.3.6 *In* $RBAC^{-}_{ST}$*:*

- *a user* $v \in U$ *is* $RBAC^{-}_{ST}$*-authorized for role* $v' \in R$ *at point* $d \in \mathcal{D}$ *if and only if there exists an au-path* $v = v_1, v_2, \ldots, v_j, \ldots, v_n = v'$ *such that* $v_j \in T$ *and* $d \in \lambda(v) \cap \lambda(v_j)$*, or there exists an au-path* $v = v_1, v_2, \ldots, v_n = v'$ *and* $d \in \lambda(v) \cap \lambda(v')$*;*

- *a role* $v \in R$ *is* $RBAC^{-}_{ST}$*-authorized for permission* $v' \in P$ *at point* $d \in \mathcal{D}$ *if and only if there exists an au-path* $v = v_1, v_2, \ldots, v_j, \ldots, v_n = v'$ *such that* $v_j \in T$ *and* $d \in \lambda(v) \cap \lambda(v_j)$*, or there exists an au-path* $v = v_1, v_2, \ldots, v_n = v'$ *and* $d \in \lambda(v) \cap \lambda(v')$*;*

- *a user* $v \in U$ *is* $RBAC^{-}_{ST}$*-authorized for permission* $v' \in P$ *at point* $d \in \mathcal{D}$ *if and only if there exists an au-path* $v = v_1, v_2, \ldots, v_j, \ldots, v_n = v'$ *such that* $v_j \in T$ *and* $d \in \lambda(v) \cap \lambda(v_j)$*, or there exists an au-path* $v = v_1, v_2, \ldots, v_i, \ldots, v_n = v'$ *such that* $v_i \in R$ *for some* i*, and* $d \in \lambda(v) \cap \lambda(v_i) \cap \lambda(v')$.

Consider Figure 4.3(f). In order to express our spatio-temporal requirements, we use RBAC^{-}_{ST} and define r' (or v) to be a trusted entity and $\lambda(r) = D$. Clearly, user v can activate roles r and r'' from any point because there exists an au-path v, r', r (and the fact that $\lambda(r) = D$ is ignored).

4.3.5 A note on RBAC$_1$-style syntax

We currently use the functions λ and μ to define the syntax of our spatio-temporal RBAC models, and a graph-based formalism to define the semantics of these models. In this section, we briefly note that we can use RBAC$_1$-style syntax to encode RBAC$^{+}_{ST}$. (It follows that RBAC$^{-}_{ST}$ and RBAC$^{=}_{ST}$ syntax can also be adjusted in the same way.)

The familiar sets and relations from the RBAC$_1$ model – U, R, P, UA, RH and PA – are adjusted to include an extra entry, corresponding to the set of points for which the entity or entity relationship is enabled. The set of users U, for example, is replaced by $U_{ST} \subseteq U \times 2^{\mathcal{D}}$; $(u, D) \in U_{ST}$ means that u is enabled for all points $d \in D$. The set of user-role assignments UA, for example, is replaced by $UA_{ST} \subseteq U \times R \times 2^{\mathcal{D}}$; $(u, r, D) \in UA_{ST}$ means that the assignment of user u to role r is enabled for all points $d \in D$. In RBAC$^{+}_{ST}$, for example, a user u may activate a role r at point d if there exist roles $r' = r_1, r_2, \ldots, r_n = r$, such that $(r_{i+1}, r_i, D_i) \in RH_{ST}$, $(u, r', D_u) \in UA_{ST}$, $i = 1, \ldots, n-1$, and $d \in D_u \cap D_1 \cap \cdots \cap D_{n-1}$.

4.3.6 Integration with ANSI-RBAC

Recall that the core and hierarchical components of ANSI-RBAC standard are defined by a set of basic element sets U, S, R and P, a set of relations UA, RH and PA, and a set of mapping functions, shown in the top part of Table 4.1.

The table demonstrates that it is easy to re-define the ANSI-RBAC functions in the context of RBAC$^{=}_{ST}$, RBAC$^{+}_{ST}$ and RBAC$^{-}_{ST}$. The function *session_users*, defined by the ANSI-RBAC standard, returns the user associated with a session, and is the same for all three models, so we omit this function in the rest of Table 4.1. Each function, when defined for RBAC$^{=}_{ST}$, RBAC$^{+}_{ST}$ and RBAC$^{-}_{ST}$, includes a parameter $d \in \mathcal{D}$. For simplicity we use our original syntax, rather than the RBAC$_1$-style syntax, for defining

ANSI-RBAC
$assigned\_users(r) = \{u \in U : (u, r) \in UA\}$
$assigned\_permissions(r) = \{p \in P : (p, r) \in PA\}$
$session\_users(s) = u$
$session\_roles(s) \subseteq \{r \in R : r \leqslant r', (session\_users(s), r') \in UA\}$
$authorized\_users(r) = \{u \in U : r \leqslant r', (u, r') \in UA\}$
$authorized\_permissions(r) = \{p \in P : r' \leqslant r, (p, r') \in PA\}$
$avail\_session\_perms(s) = \bigcup_{r \in session\_roles(s)} authorized\_permissions(r)$
$\text{RBAC}^{=}_{ST}$
$assigned\_users(r, d) = \{u \in U : (u, r) \in UA,\ d \in \lambda(u) \cap \lambda(r)\}$
$assigned\_permissions(r, d) = \{p \in P : (p, r) \in PA,\ d \in \lambda(p) \cap \lambda(r)\}$
$session\_roles(s, d) \subseteq \{r \in R : r \leqslant r', (session\_users(s), r') \in UA,\ d \in \lambda(session\_users(s)) \cap \widehat{\lambda}(r', r)\}$
$authorized\_users(r, d) = \{u \in U : r \leqslant r', (u, r') \in UA,\ d \in \lambda(u) \cap \widehat{\lambda}(r', r)\}$
$authorized\_permissions(r, d) = \{p \in P : r' \leqslant r, (p, r') \in PA,\ d \in \lambda(p) \cap \widehat{\lambda}(r, r')\}$
$avail\_session\_perms(s, d) = \bigcup_{r \in session\_roles(s,d)} authorized\_permissions(r, d)$
RBAC^{+}_{ST}
$assigned\_users(r, d) = \{u \in U : (u, r) \in UA,\ d \in \mu(u, r)\}$
$assigned\_permissions(r, d) = \{p \in P : (p, r) \in PA,\ d \in \mu(p, r)\}$
$session\_roles(s, d) \subseteq \{r \in R : r \leqslant r', (session\_users(s), r') \in UA,\ d \in \widehat{\mu}(session\_users(s), r)\}$
$authorized\_users(r, d) = \{u \in U : r \leqslant r', (u, r') \in UA,\ d \in \widehat{\mu}(u, r)\}$
$authorized\_permissions(r, d) = \{p \in P : r' \leqslant r, (p, r') \in PA,\ d \in \widehat{\mu}(r, p)\}$
$avail\_session\_perms(s, d) = \bigcup_{r \in session\_roles(s,d)} authorized\_permissions(r, d)$
RBAC^{-}_{ST}
$assigned\_users(r, d) = \{u \in U : (u, r) \in UA,\ d \in \lambda(u) \cap \lambda(r)\}$
$assigned\_permissions(r, d) = \{p \in P : (p, r) \in PA,\ d \in \lambda(p) \cap \lambda(r)\}$
$session\_roles(s, d) \subseteq \{r \in R : r \leqslant r', (session\_users(s), r') \in UA,\ d \in \lambda(session\_users(s)) \cap \lambda(r)\}$
$authorized\_users(r, d) = \{u \in U : r \leqslant r', (u, r') \in UA,\ d \in \lambda(u) \cap \lambda(r)\}$
$authorized\_permissions(r, d) = \{p \in P : r' \leqslant r, (p, r') \in PA,\ d \in \lambda(p) \cap \lambda(r)\}$
$avail\_session\_perms(s, d) = \bigcup_{r \in session\_roles(s,d)} authorized\_permissions(r, d)$

Table 4.1: Spatio-temporal ANSI-RBAC mapping functions

these functions. Note that our spatio-temporal RBAC models can be integrated quite easily with RBAC96 and the ANSI-RBAC standard.

4.4 Spatio-temporal ERBAC

In this section we extend the spatio-temporal model we have developed for RBAC_1 to include the features defined in ERBAC07.

4.4.1 ERBAC$^{=}_{ST}$: the standard model

The *standard spatio-temporal ERBAC model* (or ERBAC$^{=}_{ST}$) combines the features of RBAC$^{=}_{ST}$ and ERBAC07. In other words, we define the directed labeled graph (V, E, λ), where $E = UA \cup RH_a \cup RH_u \cup PA$.

Definition 4.4.1 *In ERBAC$^{=}_{ST}$:*

- *a user $v \in U$ may activate role $v' \in R$ at point $d \in \mathcal{D}$ if and only if there exists an a-path $v = v_1, v_2, \ldots, v_n = v'$ and $d \in \widehat{\lambda}(v, v')$;*
- *a role $v \in R$ is authorized for permission $v' \in P$ at point $d \in \mathcal{D}$ if and only if there exists a u-path $v = v_1, v_2, \ldots, v_n = v'$ and $d \in \widehat{\lambda}(v, v')$;*
- *a user $v \in U$ is authorized for permission $v' \in P$ at point $d \in \mathcal{D}$ if and only if there exists a path $v = v_1, v_2, \ldots, v_i, \ldots, v_n = v'$ such that $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an a-path, $v_i, \ldots, v_n$ is a u-path, and $d \in \widehat{\lambda}(v, v')$.*

4.4.2 ERBAC$^{+}_{ST}$: the strong model

The *strong spatio-temporal ERBAC model* (or ERBAC$^{+}_{ST}$) combines the features of RBAC$^{+}_{ST}$ and ERBAC07. In other words, we have the extended directed labeled graph (V, E, λ, μ), where $E = UA \cup RH_a \cup RH_u \cup PA$.

Definition 4.4.2 *In ERBAC$^{+}_{ST}$:*

- *a user $v \in U$ may activate role $v' \in R$ at point $d \in \mathcal{D}$ if and only if there exists an a-path $v = v_1, v_2, \ldots, v_n = v'$, and $d \in \widehat{\mu}(v, v')$;*
- *a role $v \in R$ is authorized for permission $v' \in P$ at point $d \in \mathcal{D}$ if and only if there exists a u-path $v = v_1, v_2, \ldots, v_n = v'$, and $d \in \widehat{\mu}(v, v')$;*

- *a user $v \in U$ is authorized for permission $v' \in P$ at point $d \in \mathcal{D}$ if and only if there exists a path $v = v_1, v_2, \ldots, v_i, \ldots, v_n = v'$ such that $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an a-path, $v_i, \ldots, v_n$ is a u-path, and $d \in \widehat{\mu}(v, v')$.*

4.4.3 ERBAC$^{-}_{ST}$: the weak model

The *weak spatio-temporal ERBAC model* (or ERBAC$^{-}_{ST}$) combines the features of RBAC$^{-}_{ST}$ and ERBAC07. Like ERBAC$^{=}_{ST}$, we have the extended directed labeled graph (V, E, λ), where $E = UA \cup RH_a \cup RH_u \cup PA$.

Definition 4.4.3 *In ERBAC$^{-}_{ST}$:*

- *a user $v \in U$ may activate role $v' \in R$ at point $d \in \mathcal{D}$ if and only if there exists an a-path $v = v_1, v_2, \ldots, v_n = v'$, and $d \in \lambda(v) \cap \lambda(v')$;*
- *a role $v \in R$ is authorized for permission $v' \in P$ at point $d \in \mathcal{D}$ if and only if there exists a u-path $v = v_1, v_2, \ldots, v_n = v'$, and $d \in \lambda(v) \cap \lambda(v')$;*
- *a user $v \in U$ is authorized for permission $v' \in P$ at point $d \in \mathcal{D}$ if and only if there exists a path $v = v_1, v_2, \ldots, v_i, \ldots, v_n = v'$ such that $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an a-path, $v_i, \ldots, v_n$ is a u-path, and $d \in \lambda(v) \cap \lambda(v_i) \cap \lambda(v')$.*

4.5 Practical considerations in spatio-temporal RBAC

The existence of spatio-temporal constraints will generally result in a more complex access control decision function. In this section, we consider the implementation of our spatio-temporal RBAC models in practical applications.

4.5.1 Partial transitive closure

In Section 4.3, we developed a number of spatio-temporal RBAC models that provide different authorization semantics determined by the interaction between spatio-

temporal constraints and inheritance in the RBAC model. We note that checking whether a user may activate a role or is granted a permission may be a relatively complex operation when there are spatio-temporal constraints and a role hierarchy. This is because there may be multiple paths between two roles in a role hierarchy and because we need to check whether the point at which the access request was made belongs to each of the enabling conditions on a given path. Hence, we suggest that in practical implementations, it might be useful to pre-compute the transitive closure of (part of) the RBAC96 graph.

One possibility is to construct RH^*, the transitive closure of RH, and assign $D \subseteq \mathcal{D}$ to $(r, r') \in RH^*$.

$\mathbf{RBAC}^{=}_{ST}$

In $\text{RBAC}^{=}_{ST}$, for example, this value D would be the union of the individual $\widehat{\lambda}$ values computed for each path between r and r'. That is, given $r, r' \in R$, let $\Pi(r, r')$ denote the set of paths between r and r', and for $\pi \in \Pi(r, r')$, let $\widehat{\lambda}(\pi, r, r')$ denote $\widehat{\lambda}(r, r')$ for path π. We define $\widehat{\lambda}^* : RH^* \rightarrow 2^{\mathcal{D}}$, where

$$\widehat{\lambda}^*(r, r') = \bigcup_{\pi \in \Pi(r, r')} \widehat{\lambda}(\pi, r, r')$$

Suppose, for example, that $r_4 < r_2 < r_1$ and $r_4 < r_3 < r_1$ and that r_2 and r_3 are incomparable (as shown in Figure 4.1(c)). Suppose also that $\lambda(r_i) = D_i$. Then

$$\begin{aligned}\widehat{\lambda}^*(r_1, r_4) &= (D_1 \cap D_2 \cap D_4) \cup (D_1 \cap D_3 \cap D_4) \\ &= D_1 \cap D_4 \cap (D_2 \cup D_3).\end{aligned}$$

We represent the partial transitive closure of $\text{RBAC}^{=}_{ST}$ as a tuple $(V, E^*, \lambda, \widehat{\lambda}^*)$,

where $E^* = UA \cup RH^* \cup PA$, $\lambda : V \rightarrow 2^{\mathcal{D}}$ and $\widehat{\lambda}^* : RH^* \rightarrow 2^{\mathcal{D}}$. Given $G^* = (V, E^*, \lambda, \widehat{\lambda}^*)$, a request by u to exercise a permission p at point d is granted if u has activated a role r_1 at d and there exists (r_1, r_n) and (r_n, p) in E^* such that $d \in \widehat{\lambda}^*(r_1, r_n) \cap \lambda(p)$.

$\mathbf{RBAC}^{+}_{ST}$

Similarly, in RBAC^{+}_{ST}, for $\pi \in \Pi(r, r')$, let $\widehat{\mu}(\pi, r, r')$ denote $\widehat{\mu}(r, r')$ for path π. We define $\widehat{\mu}^* : RH^* \rightarrow 2^{\mathcal{D}}$, where

$$\widehat{\mu}^*(r, r') = \bigcup_{\pi \in \Pi(r, r')} \widehat{\mu}(\pi, r, r')$$

We represent the partial transitive closure of RBAC^{+}_{ST} as a tuple $(V, E^*, \lambda, \mu, \widehat{\mu}^*)$, where $E^* = UA \cup RH^* \cup PA$. Given $G^* = (V, E^*, \lambda, \mu, \widehat{\mu}^*)$, a request by u to exercise a permission p at point d is granted if u has activated a role r_1 at d and there exists (r_1, r_n) and (r_n, p) in E^* such that $d \in \widehat{\mu}^*(r_1, r_n) \cap \mu(r_n, p)$.

$\mathbf{RBAC}^{-}_{ST}$

In RBAC^{-}_{ST}, it is not necessary to pre-compute the transitive closure of the role hierarchy. Given a RBAC^{-}_{ST} configuration $G = (V, E, \lambda)$ and a request (u, p) at point d, where $(u, r) \in E$ and $(r', p) \in E$, there might be multiple paths between two roles r and r', $\Pi(r, r')$, in the role hierarchy. However, the authorization semantics of RBAC^{-}_{ST} does not require every node on any path $\pi \in \Pi(r, r')$ to be enabled at d for granting the request (u, p). The request by u to exercise p at point d is granted by G if u has activated a role r'' at d and there exists an au-path $(r'', \ldots, p)$ and $d \in \lambda(r'') \cap \lambda(p)$.

ERBAC$^{=}_{ST}$

For the models based on ERBAC07, we compute RH^*_a, the transitive closure of RH_a, and RH^*_u, the transitive closure of RH_u. In ERBAC$^{=}_{ST}$, we define $\widehat{\lambda}^*_a : RH^*_a \to 2^{\mathcal{D}}$ and $\widehat{\lambda}^*_u : RH^*_u \to 2^{\mathcal{D}}$, where

$$\widehat{\lambda}^*_a(r, r') = \bigcup_{\pi \in \Pi(r,r')} \widehat{\lambda}(\pi, r, r') \quad \text{and} \quad \widehat{\lambda}^*_u(r, r') = \bigcup_{\pi \in \Pi(r,r')} \widehat{\lambda}(\pi, r, r')$$

We represent the partial transitive closure of ERBAC$^{=}_{ST}$ as a tuple $(V, E^*, \lambda, \widehat{\lambda}^*_a, \widehat{\lambda}^*_u)$, where $E^* = UA \cup RH^*_a \cup RH^*_u \cup PA$. Given $G^* = (V, E^*, \lambda, \widehat{\lambda}^*_a, \widehat{\lambda}^*_u)$, a request by u to exercise a permission p at point d is granted if there exists $(u, r_1), (r_1, r_i), (r_i, r_n)$ and (r_n, p) in E^* such that $d \in \lambda(u) \cap \widehat{\lambda}^*_a(r_1, r_i) \cap \widehat{\lambda}^*_u(r_i, r_n) \cap \lambda(p)$.

ERBAC$^{+}_{ST}$

Similarly, in ERBAC$^{+}_{ST}$, we define $\widehat{\mu}^*_a : RH^*_a \to 2^{\mathcal{D}}$ and $\widehat{\mu}^*_u : RH^*_u \to 2^{\mathcal{D}}$, where

$$\widehat{\mu}^*_a(r, r') = \bigcup_{\pi \in \Pi(r,r')} \widehat{\mu}(\pi, r, r') \quad \text{and} \quad \widehat{\mu}^*_u(r, r') = \bigcup_{\pi \in \Pi(r,r')} \widehat{\mu}(\pi, r, r')$$

We represent the partial transitive closure of ERBAC$^{+}_{ST}$ as a tuple $(V, E^*, \lambda, \widehat{\mu}, \widehat{\mu}^*_a, \widehat{\mu}^*_u)$, where $E^* = UA \cup RH^*_a \cup RH^*_u \cup PA$. Given $G^* = (V, E^*, \lambda, \widehat{\mu}, \widehat{\mu}^*_a, \widehat{\mu}^*_u)$, a request by u to exercise a permission p at point d is granted if there exists $(u, r_1), (r_1, r_i), (r_i, r_n)$ and (r_n, p) in E^* such that $d \in \widehat{\mu}(u, r_1) \cap \widehat{\mu}^*_a(r_1, r_i) \cap \widehat{\mu}^*_u(r_i, r_n) \cap \widehat{\mu}(r_n, p)$.

ERBAC$^{-}_{ST}$

As in RBAC$^{-}_{ST}$, it is not useful to pre-compute the transitive closure of RH_a and RH_u, because the semantics of ERBAC$^{-}_{ST}$ is only concerned with the enabling conditions on

the entities that appear in an access request.

4.5.2 Full transitive closure

$\mathbf{RBAC}^{=}_{ST}$

We now consider the full transitive closure of G, $G^\star = (V, E^\star)$, where $E^\star = (UA \cup RH \cup PA)^\star$. In $\text{RBAC}^{=}_{ST}$, given $v, v' \in V$, let $\Pi(v, v')$ denote the set of paths between v and v', and for $\pi \in \Pi(v, v')$, let $\widehat{\lambda}(\pi, v, v')$ denote $\widehat{\lambda}(v, v')$ for path π. We define $\widehat{\lambda}^\star : E^\star \to 2^{\mathcal{D}}$, where

$$\widehat{\lambda}^\star(v, v') = \bigcup_{\pi \in \Pi(v, v')} \widehat{\lambda}(\pi, v, v')$$

We represent the full transitive closure of $\text{RBAC}^{=}_{ST}$ as a tuple $(V, E^\star, \lambda, \widehat{\lambda}^\star)$. Given $G^\star = (V, E^\star, \lambda, \widehat{\lambda}^\star)$, a request by u to exercise permission p at point d is granted if there exists (u, p) in $E^\star$ such that $d \in \widehat{\lambda}^\star(u, p)$.

$\mathbf{RBAC}^{+}_{ST}$

Similarly, in RBAC^{+}_{ST}, for $\pi \in \Pi(v, v')$, let $\widehat{\mu}(\pi, v, v')$ denote $\widehat{\mu}(v, v')$ for path π. We define $\widehat{\mu}^\star : E^\star \to 2^{\mathcal{D}}$, where

$$\widehat{\mu}^\star(v, v') = \bigcup_{\pi \in \Pi(v, v')} \widehat{\mu}(\pi, v, v')$$

We represent the full transitive closure of RBAC^{+}_{ST} as a tuple $(V, E^\star, \lambda, \mu, \widehat{\mu}^\star)$. Given $G^\star = (V, E^\star, \lambda, \mu, \widehat{\mu}^\star)$, a request by u to exercise permission p at point d is granted if there exists (u, p) in $E^\star$ such that $d \in \widehat{\mu}^\star(u, p)$.

$\mathbf{ERBAC}^{=}_{ST}$

For the spatio-temporal ERBAC07 models, we consider the full transitive closure of G, $G^\star = (V, E^\star)$, where $E^\star = E^\star_a \cup E^\star_u$, $E^\star_a = (UA \cup RH_a)^\star$ and $E^\star_u = (RH_u \cup PA)^\star$. In

ERBAC$^{=}_{ST}$, we define $\widehat{\lambda}^{\star}_{a} : E^{\star}_{a} \rightarrow 2^{\mathcal{D}}$ and $\widehat{\lambda}^{\star}_{u} : E^{\star}_{u} \rightarrow 2^{\mathcal{D}}$, where

$$\widehat{\lambda}^{\star}_{a}(v, v') = \bigcup_{\pi \in \Pi(v,v')} \widehat{\lambda}(\pi, v, v') \quad \text{and} \quad \widehat{\lambda}^{\star}_{u}(v, v') = \bigcup_{\pi \in \Pi(v,v')} \widehat{\lambda}(\pi, v, v')$$

We represent the full transitive closure of ERBAC$^{=}_{ST}$ as a tuple $(V, E^{\star}, \lambda, \widehat{\lambda}^{\star}_{a}, \widehat{\lambda}^{\star}_{u})$. Given $G^{\star} = (V, E^{\star}, \lambda, \widehat{\lambda}^{\star}_{a}, \widehat{\lambda}^{\star}_{u})$, a request by u to exercise permission p at point d is granted if there exists (u, r) and (r, p) in $E^{\star}$ such that $d \in \widehat{\lambda}^{\star}_{a}(u, r) \cup \widehat{\lambda}^{\star}_{u}(r, p)$.

ERBAC$^{+}_{ST}$

Similarly, in ERBAC$^{+}_{ST}$, we define $\widehat{\mu}^{\star}_{a} : E^{\star}_{a} \rightarrow 2^{\mathcal{D}}$ and $\widehat{\mu}^{\star}_{u} : E^{\star}_{u} \rightarrow 2^{\mathcal{D}}$, where

$$\widehat{\mu}^{\star}_{a}(v, v') = \bigcup_{\pi \in \Pi(v,v')} \widehat{\mu}(\pi, v, v') \quad \text{and} \quad \widehat{\mu}^{\star}_{u}(v, v') = \bigcup_{\pi \in \Pi(v,v')} \widehat{\mu}(\pi, v, v')$$

We represent the full transitive closure of ERBAC$^{+}_{ST}$ as a tuple $(V, E^{\star}, \lambda, \mu, \widehat{\mu}^{\star}_{a}, \widehat{\mu}^{\star}_{u})$. Given $G^{\star} = (V, E^{\star}, \lambda, \mu, \widehat{\mu}^{\star}_{a}, \widehat{\mu}^{\star}_{u})$, a request by u to exercise permission p at point d is granted if there exists (u, r) and (r, p) in $E^{\star}$ such that $d \in \widehat{\mu}^{\star}_{a}(u, r) \cup \widehat{\mu}^{\star}_{u}(r, p)$.

Similarly, it is not useful to pre-compute the full transitive closure of RBAC$^{-}_{ST}$ and ERBAC$^{-}_{ST}$. Note that computing the full transitive closure will only be practical for relatively small numbers of users and permissions, so it is likely that computing the partial transitive closure will be more useful in practice.

4.5.3 Is the use of hierarchies realistic?

The examples in Figure 4.3 illustrate that the presence of a role hierarchy significantly complicates the specification of spatio-temporal constraints. We argued in Section 4.3 that there were at least three different models that could be used; even then, it was necessary to introduce the notion of trusted entities for certain scenarios. This suggests that there are many possible encodings of spatio-temporal restrictions in the presence

of a role hierarchy. Choosing the appropriate model may well be difficult, and encoding the desired enterprise security policies within such a model is also likely to be non-trivial. In the next two sections, we consider two simple strategies that might be used to mitigate these difficulties.

Flat spatio-temporal RBAC

In practice, it might well be preferable to assume that the set of roles is unordered, as in core ANSI-RBAC standard or flat RBAC96 (RBAC_0). This means, of course, that the number of user- and permission-role assignments will increase (because such assignments are often implicitly generated by assignments to other roles in the presence of a role hierarchy). However, flat RBAC^{+}_{ST} can be used to specify most spatio-temporal constraints.

Consider the spatio-temporal requirements for the configuration of RBAC in Figure 4.3(f) on Page 95. We transform the RBAC_1 configuration to flat RBAC as follows: $U = \{u, v\}$, $R = \{r, r', r''\}$ and $UA = \{(v, r'), (v, r), (v, r''), (u, r), (u, r'')\}$. We only need to define $\mu(u, r) = D$; all other nodes and edges are enabled for any $d \in \mathcal{D}$. In fact, we can encode the spatio-temporal requirements for other configurations of RBAC in Figure 4.3 using flat RBAC^{+}_{ST}.

To formally illustrate the expressive power of flat RBAC^{+}_{ST}, we now prove how to transform hierarchical spatio-temporal RBAC, RBAC^{+}_{ST}, $\text{RBAC}^{=}_{ST}$, and RBAC^{-}_{ST}, configurations into an equivalent flat RBAC^{+}_{ST} configuration, where equivalence means that the same set of requests is authorized by both configurations.

Theorem 4.5.1 *Let $\Sigma = (UA, RH, PA, \lambda, \mu)$ define an $RBAC^{+}_{ST}$ system. Then there exists a flat $RBAC^{+}_{ST}$ system $\Sigma' = (UA', PA', \lambda', \mu')$ such that user u is authorized for permission p at point $d \in \mathcal{D}$ in Σ if and only if u is authorized for p at point $d \in \mathcal{D}$ in*

Σ'.

Proof Given $\Sigma = (UA, RH, PA, \lambda, \mu)$, we construct a flat RBAC^{+}_{ST} system $\Sigma' = (UA', PA', \lambda', \mu')$ using the following procedure.

1. Construct the full transitive closure of the RBAC^{+}_{ST} system, $\Sigma^{\star} = (E^{\star}, \lambda, \mu, \widehat{\mu}^{\star})$, where $E^{\star} = (UA \cup RH \cup PA)^{\star}$ and $\widehat{\mu}^{\star} : E^{\star} \rightarrow 2^{\mathcal{D}}$;

2. For all $u \in U$ and for all $r \in R$ such that $(u, r) \in E^{\star}$, we define $(u, r) \in UA'$ and $\mu'(u, r) = \widehat{\mu}^{\star}(u, r)$;

3. For all $p \in P$ and for all $r \in R$ such that $(r, p) \in E^{\star}$, we define $(p, r) \in PA'$ and $\mu'(r, p) = \widehat{\mu}^{\star}(r, p)$.

4. For all $v \in V = (U \cup R \cup P)$, we define $\lambda'(v) = \lambda(v)$.

We now prove the result about equivalence. If u is authorized for p at d in Σ, then by Definition 4.3.2, there exists $r \in R$ such that $u, \ldots, r$ is au-path, $r, \ldots, p$ is au-path and $d \in \widehat{\mu}(u, r) \cap \widehat{\mu}(r, p)$ in Σ. By Step 1 and 2, $(u, r) \in UA'$ and $\mu'(u, r) \supseteq \widehat{\mu}(u, r)$. By Step 1 and 3, $(p, r) \in PA'$ and $\mu'(r, p) \supseteq \widehat{\mu}(r, p)$. Hence, there exists r such that $(u, r) \in UA'$, $(p, r) \in PA'$ and $d \in \mu'(u, r) \cap \mu'(r, p)$. Again, by Definition 4.3.2, u is authorized for p at d in Σ'.

Conversely, if u is authorized for p at d in Σ', then there exists $r \in R$ such that $(u, r) \in UA'$, $(p, r) \in PA'$ and $d \in \mu'(u, r) \cap \mu'(r, p)$ in Σ'. By Steps 1, 2 and 3, there exists an au-path from u to r and an au-path from r to p such that $d \in \widehat{\mu}(u, r) \cap \widehat{\mu}(r, p)$. Then u is authorized for p at d in Σ. ■

Corollary 4.5.1 *Let $\Sigma = (UA, RH, PA, \lambda)$ define an $RBAC^{\leftrightharpoons}_{ST}$ system. Then there exists a flat $RBAC^{+}_{ST}$ system $\Sigma' = (UA', PA', \lambda', \mu')$ such that user u is authorized for permission p at point $d \in \mathcal{D}$ in Σ if and only if u is authorized for p at point $d \in \mathcal{D}$ in Σ'.*

Proof Given $\Sigma = (UA, RH, PA, \lambda)$, we construct an equivalent RBAC^{+}_{ST} system $\Sigma'' = (UA, RH, PA, \lambda, \mu)$, where for all $(v, v') \in E = UA \cup RH \cup PA$, $\mu(v, v') = \lambda(v) \cap \lambda(v')$. Then by Theorem 4.5.1, there exists a flat RBAC^{+}_{ST} system $\Sigma' = (UA', PA', \lambda', \mu')$ that is equivalent to Σ''. Hence, by transitivity, user u is authorized for p at d in Σ if and only if u is authorized for p at d in Σ'. ■

Theorem 4.5.2 *Let $\Sigma = (UA, RH, PA, \lambda)$ define an $RBAC^{-}_{ST}$ system. Then there exists a flat $RBAC^{+}_{ST}$ system $\Sigma' = (UA', PA', \lambda', \mu')$ such that user u is authorized for permission p at point $d \in \mathcal{D}$ in Σ if and only if u is authorized for p at point $d \in \mathcal{D}$ in Σ'.*

Proof Given $\Sigma = (UA, RH, PA, \lambda)$, we construct a flat RBAC^{+}_{ST} system $\Sigma' = (UA', PA', \lambda', \mu')$ using the following procedure.

1. Construct the full transitive closure of the RBAC^{-}_{ST} system, $\Sigma^{\star} = (E^{\star}, \lambda)$, where $E^{\star} = (UA \cup RH \cup PA)^{\star}$;

2. For all $u \in U$ and for all $r \in R$ such that $(u, r) \in E^{\star}$, we define $(u, r) \in UA'$ and $\mu'(u, r) = \lambda(u) \cap \lambda(r)$;

3. For all $p \in P$ and for all $r \in R$ such that $(r, p) \in E^{\star}$, we define $(p, r) \in PA'$ and $\mu'(r, p) = \lambda(r) \cap \lambda(p)$;

4. For all $v \in V = (U \cup R \cup P)$, we define $\lambda'(v) = \lambda(v)$.

We now prove the result about equivalence. If u is authorized for p at d in Σ, then by Definition 4.3.3, there exists $r \in R$ such that $u, \ldots, r, \ldots, p$ is au-path, and $d \in \lambda(u) \cap \lambda(r) \cap \lambda(p)$ in Σ. By Step 1 and 2, $(u, r) \in UA'$ and $\mu'(u, r) = \lambda(u) \cap \lambda(r)$. By Step 1 and 3, $(p, r) \in PA'$ and $\mu'(r, p) = \lambda(r) \cap \lambda(p)$. Hence, there exists r such that $(u, r) \in UA'$ and $(p, r) \in PA'$, and $d \in \mu'(u, r) \cap \mu'(r, p)$. Again, by Definition 4.3.2, u is authorized for p at d in Σ'.

If u is authorized for p at d in Σ', then there exists $r \in R$ such that $(u, r) \in UA'$ and $(p, r) \in PA'$, and $d \in \mu'(u, r) \cap \mu'(r, p)$ in Σ'. By Steps 1, 2 and 3, there exists an au-path from u to r and an au-path from r to p such that $d \in \lambda(u) \cap \lambda(r)$ and $d \in \lambda(r) \cap \lambda(p)$. In other words, there exists an au-path $u, \ldots, r, \ldots, p$ and $d \in \lambda(u) \cap \lambda(r) \cap \lambda(p)$ in Σ. Then u is authorized for p at d in Σ. ■

We have shown that any spatio-temporal constraints that encoded in hierarchical spatio-temporal RBAC models ($\text{RBAC}^{=}_{ST}$, RBAC^{+}_{ST} and RBAC^{-}_{ST}) can be also encoded using the flat RBAC^{+}_{ST} model. Note that the authorization semantics of $\text{RBAC}^{=}_{ST}$ is exactly same as those of RBAC^{-}_{ST} when the role hierarchy is empty, hence we only consider a single model (flat $\text{RBAC}^{=}_{ST}$) for defining constraints on the entities of RBAC_0. Furthermore, flat $\text{RBAC}^{=}_{ST}$ is a special case of flat RBAC^{+}_{ST}, where the enabling conditions of every edge is defined to a set of points at which both end points are enabled. Figure 4.4 shows a hierarchy that summarizes the comparison of spatio-temporal models to encode spatio-temporal constraints. For example, we can see that the most powerful model is flat RBAC^{+}_{ST} that can specify various spatio-temporal constraints that encoded by its junior models.

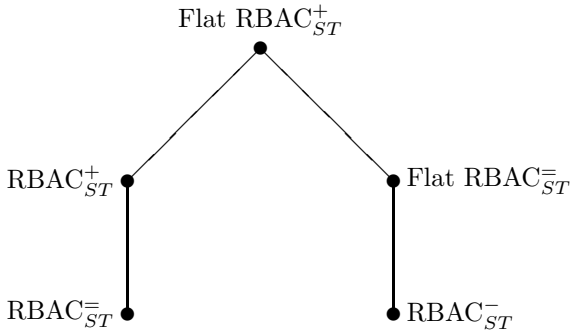


Figure 4.4: A comparison of spatio-temporal models for encoding constraints

Eliminate enabling restrictions on roles and the role hierarchy

We now propose an alternative approach that is distinct from the strategy advocated in the previous section by including a role hierarchy and eliminating enabling restrictions on roles and the role hierarchy: that is, set $\mu(r, r') = \mathcal{D}$ for all $r, r' \in R$. Since $\mu(r, r') \subseteq \lambda(r) \cap \lambda(r')$, $\lambda(r) = \lambda(r') = \mathcal{D}$ for all $r, r' \in R$. In other words, all roles and their associated edges are enabled at all points in the spatial-temporal domain, and restrictions are only imposed at the outer nodes (users and permissions) and edges (user-role and permission-role assignments) of the RBAC graph. This approach is completely contrary to existing approaches, in which roles are usually the only entities for which such enabling conditions are defined (see [13], for example).

An example that is often quoted in the temporal RBAC literature is that of a `night-doctor` role, which should only be enabled during the night shift hours [13]. We would argue that instead of imposing the enabling condition on the `night-doctor` role, we should impose the condition on any assignment of that role to a user. This does not preclude the same user from also being assigned to the `day-doctor` role (which would have a different enabling condition on the user-role assignment).

It would not be difficult to implement this kind of approach. Let us assume that we have a `night-doctor` role, which should only be activated during the night shift. Then, whenever a user is assigned to this role, an enabling condition is automatically generated for that user-role assignment. (If the intersection of the user's enabling condition and this condition is empty, then the assignment fails.)

Let us now consider the impact of setting $\mu(r, r') = \mathcal{D}$ for all $r, r' \in R$ in each of the three spatio-temporal RBAC models. In $\text{RBAC}^{=}_{ST}$, a user u may activate a role r at point d if there is an au-path $u, r_1, \ldots, r_n = r$ and $d \in \lambda(u)$, a role r is authorized for permission p at point d if there is an au-path $r = r_1, \ldots, r_n, p$ and

$d \in \lambda(p)$, and user u is authorized for permission p at point d if there is an au-path $u, r_1, \ldots, r_n, p$ and $d \in \lambda(u) \cap \lambda(p)$. Note that the authorization semantics of $\text{RBAC}^{=}_{ST}$ and RBAC^{-}_{ST} coincide with this restriction, which means administrators have fewer options of encoding spatio-temporal constraints by using this approach.

In RBAC^{+}_{ST}, a user u may activate a role r at point d if there is an au-path $u, r_1, \ldots, r_n = r$ and $d \in \mu(u, r_1)$, a role r is authorized for permission p at point d if there is an au-path $r = r_1, \ldots, r_n, p$ and $d \in \mu(r_n, p)$, and user u is authorized for permission p at point d if there is an au-path $u, r_1, \ldots, r_n, p$ and $d \in \mu(u, r_1) \cap \mu(r_n, p)$.

4.5.4 Concluding remarks

We have developed three spatio-temporal RBAC models and introduced the notion of trusted entities to specify spatio-temporal requirements in different configurations of the RBAC_1 model. We also extended our models to include spatio-temporal requirements for ERBAC07. All these models study different interactions between a role hierarchy and spatio-temporal constraints, which might give rise to complex computations when checking access requests. We suggest that it is appropriate to pre-compute spatio-temporal constraints over the transitive closure of the role hierarchy to improve the efficiency of access request checking. On the other hand, it is unlikely that it is useful to pre-compute spatio-temporal constraints over the full transitive closure of the RBAC_1 graph in many practical systems, because the size of $E^{\star}$ will be very large. However, deciding access requests can be performed far more quickly than in existing approaches [58].

On the other hand, the need for different spatio-tepmoral models arises because once enabling conditions are imposed on roles in the presence of role hierarchy, there are a number of different choices for the semantics of authorization. In practice, it is complicated and error-prone to specify comprehensive spatio-temporal requirements

in a hierarchical RBAC model. Therefore, we proposed two approaches to address these difficulties: specifying spatio-temporal constraints on flat RBAC, and eliminating spatio-temporal constraints on roles and the role hierarchy.

We would argue that, in many practical situations, the most appropriate approach is to use flat RBAC^{+}_{ST} to specify spatio-temporal constraints. We have illustrated that most hierarchical spatio-temporal RBAC configurations, such as $\text{RBAC}^{=}_{ST}$, RBAC^{+}_{ST} and RBAC^{-}_{ST} can be transformed into a equivalent flat RBAC^{+}_{ST} configuration in terms of what requests are authorized.

However, when there are very large numbers of user and permissions, it may well be appropriate to use role hierarchies, thereby avoiding large numbers of user- and permission-role assignments. In this case, it may be appropriate to set $\mu(r, r') = \mathcal{D}$ for all $r, r' \in R$, and specify enabling conditions on restrictions on outer nodes and edges, such as users and user-role assignments, of the RBAC graph. We should perhaps note that the underlying "philosophy" of RBAC is to use roles to reduce the burden of administration, and that our suggestion of applying enabling constraints to users and user-role assignment is inconsistent with this basic tenet. As we have seen, however, many situations may require constraints on users and user-role assignment, rather than roles. This suggests that incorporating spatio-temporal constraints within RBAC is likely to require some trade-off between the complexity of policies that can be supported and the complexity of constraint specification and administration.

4.6 Spatio-temporal domains

Much of the work in extending RBAC to include spatial and temporal restrictions on entities and entity relationships has spent a considerable amount of time on how these restrictions might be specified. The authors of GTRBAC, for example, define

a syntax for temporal restrictions using the notion of *calendars* [68]. Although we believe that it is of much greater importance to understand the interaction between RBAC inheritance and such restrictions, we now briefly consider how sets of points within a spatio-temporal domain might be specified.

Broadly speaking, there are two possibilities: *concrete* and *symbolic* domains. A concrete domain makes use of actual points in space-time, whereas a symbolic domain uses labels as synonyms for sets of points in an associated concrete domain. We consider spatial and temporal domains separately. A single spatio-temporal domain $\mathcal{D}$ can be treated as a pair $(\mathcal{S}, \mathcal{T})$, where $\mathcal{S}$ is a spatial domain and $\mathcal{T}$ is a temporal domain.

4.6.1 Representing location

A concrete spatial domain is defined by a co-ordinate system: we could use standard Euclidean space or we may use spherical or cylindrical co-ordinate systems, for example. The system chosen will be entirely dependent on the method by which user location is determined. For ease of exposition, we will define the concrete spatial domain to be $\mathcal{S} = \{(x, y) : x, y \in \mathbb{Z}\}$. In other words, points in space are defined by two integer co-ordinates.

An *atomic location* is defined to be a rectangle, which is defined by the co-ordinates of its lower-left and upper-right corners.[6] That is, a rectangle is a pair $[l, r]$, where $l, r \in \mathcal{S}$. A *location* is the union of one or more disjoint atomic locations: clearly, the set of locations is a subset of $2^{\mathcal{S}}$ and λ maps an entity to a location.

Having defined a concrete spatial domain, we may define a symbolic spatial domain, in which locations are associated with labels. Symbolic locations may be defined to be the union of other symbolic locations; these symbolic locations may overlap. Having

[6] Of course, we could define a location to be a circular region in the concrete spatial domain, by defining the center $c \in \mathcal{S}$ and radius $r \in \mathbb{Z}$ of the circle. Again, the definition of location will be determined by the method used to identify the position of a user.

defined a set of symbolic locations, we must define a mapping from the set of symbolic locations to concrete locations. We may also use λ to map entities to symbolic locations, and then map the symbolic location to a concrete location.

Let $s \in \mathcal{S}$ be a point in the concrete spatial domain, and let $L \subseteq \mathcal{S}$ be a concrete location. We write $s \in L$ if s belongs to one of the atomic locations contained in L. If L is a union of symbolic locations, we write $s \in L$ to denote that s belongs to at least one of the symbolic locations contained in L.

4.6.2 Representing time

We assume the existence of a clock, whose ticks are indexed by the natural numbers $\mathbb{N}$.[7] An *atomic interval* in the concrete temporal domain $\mathcal{T} = \mathbb{N}$, is defined by a start point $t_1 \in \mathcal{T}$ and an end point $t_2 \in \mathcal{T}$, and written as $[t_1, t_2]$. An *interval* is defined to be the union of one or more disjoint atomic intervals; λ maps an entity to an interval.

We may also define a symbolic temporal domain, in which intervals are associated with labels. We could, for example, define the symbolic intervals `21:August:2007`, `Mondays:2007`, `WorkingHours` etc. We may use λ to map entities to symbolic intervals.

Let $t \in \mathcal{T}$ be a point in the concrete temporal domain, and let $I \subseteq \mathcal{T}$ be a concrete interval. We write $t \in I$ if t belongs to one of the atomic intervals contained in I. If I is the union of symbolic intervals, we write $t \in I$ to denote that t belongs to at least one of the symbolic intervals contained in I.

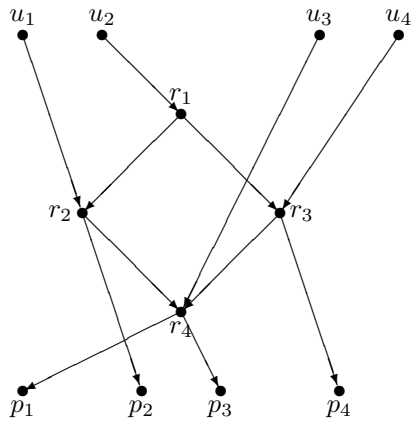
4.6.3 Example

In this section we present examples to illustrate the applications of spatial RBAC$^{=}_{ST}$, temporal RBAC$^{=}_{ST}$ and spatio-temporal ERBAC$^{+}_{ST}$ in a practical environment.

[7] It should be noted that representing time will be more complex than this for many applications; typically a local time is relative to a location and a time of year. Our representation of time, as for location, is merely illustrative.

u_1	Alice
u_2	Bob
u_3	Chris
u_4	Diane
r_1	Head of department
r_2	Academic staff
r_3	Admin staff
r_4	Student
p_1	Access resources via Metalib
p_2	Access ACM and IEEE library
p_3	Listen to presentations
p_4	View staff profile

(a) RBAC_1 entities



(b) Graphical representation of RBAC_1 relations

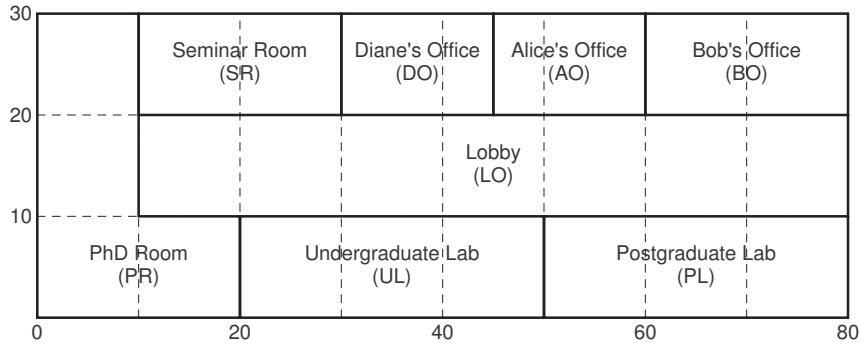
Figure 4.5: An example of an RBAC_1 configuration

Spatial $\text{RBAC}^{=}_{ST}$

Figures 4.5 and 4.6 illustrates some of the concept that have been introduced in this chapter. Figure 4.5(a) lists a number of RBAC entities associated with a computer science department at a university. Figure 4.5(b) illustrates the relationships between these entities. A user u_2, who is assigned to role r_1, is allowed to activate roles r_2, r_3, r_4 in any session. In RBAC_1, u_2 is authorized to invoke permissions p_1, p_2, p_3, p_4 since any permission can be reached by u_2 via a path in the graph.

In order to define spatial constraints for this example, we describe the layout of a floor in a university computer building (CB), as shown in Figure 4.6(a). Figure 4.6(b) defines enabling constraints for the RBAC entities in Figure 4.5(a).[8] For example, permission to access the ACM and IEEE libraries (p_2) is only allowed if the requester is in the seminar room (SR), Alice's office (AO), or Bob's office (BO). In Diane's office, for example, permissions p_2 and p_3 are not enabled; however, Diane is allowed to activate r_3 (Admin staff), thereby enabling her to view staff profile.

[8]Note that all roles are enabled everywhere within the computer building, following the approach suggested in Section 4.5.3.



(a) Spatial domain

	Spatial domain		**Temporal domain**
Entity	**Symbolic**	**Concrete**	**Symbolic**
u_1	CB	[(0,0),(80,30)]	09:00-17:59
u_2	CB	[(0,0),(80,30)]	09:00-17:59
u_3	CB	[(0,0),(80,30)]	Always
u_4	CB	[(0,0),(80,30)]	Always
r_1	CB	[(0,0),(80,30)]	Always
r_2	CB	[(0,0),(80,30)]	Always
r_3	CB	[(0,0),(80,30)]	Always
r_4	CB	[(0,0),(80,30)]	Always
p_1	CB	[(0,0),(80,30)]	Always
p_2	SR $\cup$ AO $\cup$ BO	[(10,20),(30,30)] $\cup$ [(45,20),(80,30)]	09:00-17:59
p_3	SR	[(10,20),(30,30)]	12:00-13:00
p_4	DO	[(30,20),(45,30)]	Always

(b) Spatial-temporal constraints on nodes

$\mu((u_1, r_2))$	(CB, 09:00$-$13:00 $\cup$ 14:00$-$17:59)
$\mu((u_2, r_1))$	(CB, 09:00$-$13:00 $\cup$ 14:00$-$17:59)
$\mu((u_3, r_4))$	(CB, Always)
$\mu((u_4, r_3))$	(CB, Always)
$\mu((r_1, r_2))$	(CB, Always)
$\mu((r_1, r_3))$	(CB, Always)
$\mu((r_2, r_4))$	(CB, Always)
$\mu((r_3, r_4))$	(CB, Always)
$\mu((r_4, p_1))$	(CB, Always)
$\mu((r_2, p_2))$	(SR $\cup$ AO $\cup$ BO, 09:00$-$13:00 $\cup$ 14:00$-$17:59)
$\mu((r_4, r_3))$	(SR, 12:00$-$13:00)
$\mu((r_3, p_4))$	(DO, Always)

(c) Spatio-temporal constraints on edges

Figure 4.6: An example of the specification of spatio-temporal domains

Temporal RBAC$^{=}_{ST}$

Let us consider the graphical formulation of RBAC_1 configurations for the computer building shown in Figures 4.5(a) and 4.5(b). Let us assume that Figure 4.6(b) represents symbolic temporal domains for all entities of RBAC_1 in the computer building example. Then at a particular point of time 14:00, the permission p_3 is not enabled. All other entities are enabled, and related edges exist at time 14:00. For example, Alice is allowed to activate role r_2 to use the permission p_1 that is inherited from role r_4 at time instant 14:00.

Spatio-temporal ERBAC$^{+}_{ST}$

Consider the activation and usage hierarchies of ERBAC07 shown in Figures 4.7(a) and 4.7(b), respectively, and the user-role assignment and the permission-role assignment are as same as the configurations in Figure 4.5(b). For example, user u_2 is authorized to activate role r_1, but is not thereby authorized for permission p_4 which is not inherited by r_1 in the permission usage hierarchy. Let us assume that Figure 4.5(a) represents ERBAC07 entities in the computer building.

Figures 4.6(b) and 4.6(c) represent the spatio-temporal enabling conditions for ERBAC07 entities and relations. Note that a user must explicitly activate the Admin staff role in order to use the permissions associated with this role. Note also that the specification of spatio-temporal domains on edges observes the consistency constraint between nodes and edges. At a particular spatio-temporal point (Alice's office, 13:30), Alice can not activate the role (Academic staff), because Alice is not assigned academic staff role at point (AO,13:30) although both user (Alice) and role (Academic staff) are enabled at point (AO,13:30). On the other hand, at point (Diane's office, 14:00), Bob can activate the role r_3 (Admin staff) to use the permission p_4 (View staff profile).

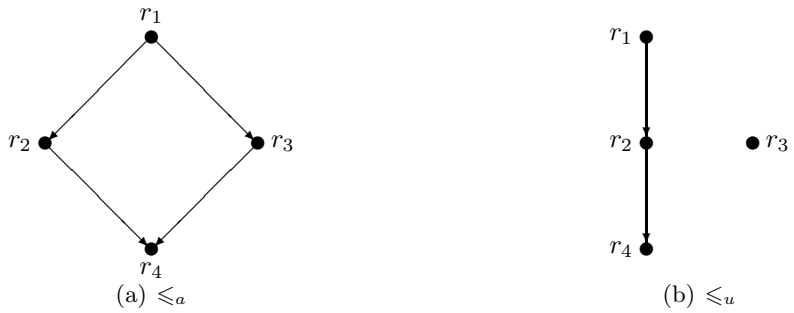


Figure 4.7: An example of ERBAC07 activation and usage hierarchies

4.7 Related work

In this section, we review some related work on context-based access control. In particular, we examine the temporal constraints in the GTRBAC model and the spatio-temporal RBAC model of Ray and Toahchoodee [76] in more detail. We explain why we believe that our spatio-temporal models are more attractive than related work according to several criteria: well-defined authorization semantics, syntactic completeness (constraints on all RBAC entities and relations), consistency (absence of conflicts, resolution of conflicts), and syntactic simplicity (number of predicates or functions).

Work has been done on spatial constraints in the context of mandatory access control (MAC) [75], discretionary access control (DAC) [4] and RBAC models [14, 49]. This work has either studied spatial constraints in traditional access control models [4, 75], rather than RBAC, or proposed a limited spatially constrained RBAC [49]. GEO-RBAC [14] introduces a comprehensive spatial RBAC model for specifying spatial constraints on roles and treats locations as objects in the RBAC model. The GEO-HRBAC model defines the role hierarchy based on the containment of locations. Compared with our models, we believe that GEO-HRBAC is too application-dependent, and focuses on controlling access on different locations.

There has also been research on more general contextual information to achieve fined-grained role-based access control. The teaM-based access control (TMAC) ap-

proach extends RBAC with the notion of team and context-based permission activation [43, 87]. Covington *et al* [28] introduce the concept of environment roles in RBAC which are activated based on the values of environmental conditions. Strembeck *et al* [86] introduce the concept of context constraints in RBAC which is used to restrict usage of permissions through considering environmental factors in access control decision. Although all above works attempted to incorporate general contextual information in RBAC model, none of them has comprehensively studied the impacts of context on all the components of the RBAC model. We now consider the two most significant pieces of work on spatio-temporal RBAC.

4.7.1 Temporal constraints in GTRBAC

The temporal-RBAC model (TRBAC) introduces temporal constraints which limit the time during which a role is enabled and activated [13]. Generalized TRBAC (GTRBAC) is an extension of TRBAC that applies temporal constraints to the assignment of users and permissions to roles [58]. GTRBAC does not consider temporal constraints on users (sessions), permissions and role hierarchical relationships. Moreover, GTRBAC, unlike our models, does not impose any consistency constraints on the user- and permission-role assignments and role-role relationships.

In addition to defining the hybrid role hierarchy that contains the role activation hierarchy $\leqslant_a$, permission-usage hierarchy $\leqslant_u$ and permission-activation hierarchy $\leqslant$, GTRBAC further sub-divides hierarchies into "weakly" and "strongly" restricted; the authorization semantics for these hierarchies differ. The weakly restricted semantics

for permission usage [58, Table 7], are defined by

$$\texttt{can\_be\_acquired}(p, x, t) \leftarrow \forall p,\ (x \geqslant_u y) \wedge \texttt{enabled}(x, t) \wedge \texttt{can\_be\_acquired}(p, y, t). \tag{4.1}$$

The intuition seems to be that if x is enabled, $x \geqslant_u y$ and y can acquire permission p, then x can acquire permission p. To quote Joshi *et al*: "The weakly restricted hierarchies allow inheritance or activation semantics in the nonoverlapping intervals... only role x needs to be enabled at time t for the [usage] inheritance semantics to apply".

However, there are a number of problems with this definition. The predicate `can_be_acquired` is defined recursively, but there is no base case; in particular, replacing x with y (which is legitimate, since $y \geqslant y$) in the rule above means we have a circular definition. Presumably the base case is that $(p, y) \in PA$, but the presence of the parameter t in `can_be_acquired` suggests that there may be an enabling condition on this assignment. Similar problems exist for weakly restricted semantics for role activation, and for strongly restricted semantics for permission acquisition and role activation.

Without a base case, it is impossible to determine the intended meaning of weakly and strongly restricted hierarchies. Moreover, it seems that any enabling conditions on roles between x and y are ignored. This makes a direct comparison between our models and GTRBAC impossible. The strongly restricted semantics require x and y to be enabled, which suggests that strongly restricted semantics in GTRBAC are (intended to be) somewhat similar to RBAC^{-}_{ST}.

4.7.2 Spatio-temporal RBAC

Ray and Toahchoodee developed a spatio-temporal RBAC model [76] that is strongly influenced by GTRBAC. Indeed, the main novelty of their approach is to introduce spatial and temporal constraints on all the components of RBAC. They also consider the consistency of the constraints on user-role and permission-role assignments.

Like ERBAC07, they introduce a role activation hierarchy $\leqslant_a$ and a permission usage hierarchy $\leqslant_u$. They also define temporal constraints, location constraints, and temporal and location constraints on these two role hierarchies. Let us consider the representative example of "time location restricted permission inheritance hierarchy" [76, Definition 13], where

$$\texttt{PermRoleAcquire}(p, x, d, l) \leftarrow \forall p, (x \geqslant_u y) \wedge \texttt{PermRoleAcquire}(p, y, d, l). \tag{4.2}$$

Here, d represents a set of time points and l a set of points in space. Again, it is not clear what the base case is, and intermediate roles between x and y are ignored.

In addition, this definition may give rise to conflicts within the specification of enabling conditions. If $\texttt{PermRoleAcquire}(p, r, d, l)$ holds then r and p are enabled at all points within d and l [76, Section 4.5]. Now let us assume that

- $\texttt{RoleEnableLoc}(x) = l'$ (x is enabled at l') and $\texttt{RoleEableDur}(x) = d'$ (x is enabled during d'),
- $\texttt{PermRoleAcquire}(p, y, d, l)$ holds and $x \geqslant_u y$,
- $d' \subset d$ and $l' \subset l$.

Then we have $\texttt{PermRoleAcquire}(p, x, d, l)$, by (4.2). This implies that x is enabled at $l \supset l'$ and $d \supset d'$, which contradicts the enabling conditions defined on x. Similar conflicts exist for weakly temporal and location restricted permission acquisition.

4.7.3 Summary

We conclude that despite the considerable amount of research on spatio-temporal RBAC models, existing work suffers from significant shortcomings. These include poorly defined authorization semantics, syntax that is both complicated and inadequate, lack of compatibility with RBAC96/ANSI-RBAC standard and a lack of consistency. The GTRBAC model and that of Ray and Toahchoodee – perhaps the two most detailed models in the literature – suffer from all of these problems. We have already noted some of these problems in earlier sections. Comparing the syntactic complexity, Joshi *et al* define 23 predicates in GTRBAC, Ray and Toahchoodee define 16, whereas we supplement RBAC96 with just two functions λ and μ. Perhaps the biggest difference between our approach and existing work is to focus on semantics, rather than syntax; we believe the former to be much the harder and less well understood of the two aspects of a spatio-temporal RBAC model.

4.8 Conclusion

In this chapter, we developed the ERBAC07 model based on RBAC_1, and influenced by existing models such as ERBAC96 and GTRBAC. We constructed a number of spatio-temporal role-based models based on RBAC_1 and ERBAC07 using a simple extension of the syntax used for RBAC_1. We introduced a graph-based formalism to explain the semantics of RBAC_1 and ERBAC07, and used this as a basis for defining the semantics of our spatio-temporal models. We note, in passing, that these semantics might be a useful addition to the ANSI-RBAC standard.

We examined the difficulties that arise when enabling constraints are placed on roles in the presence of a role hierarchy, not only on the complexity of evaluating access requests, but also on the complexity of defining spatio-temporal RBAC policies. We

proposed that some pre-computation of enabling conditions on the transitive closure of (part of) the RBAC graph can be performed to simplify the evaluation of access requests when there are requirements for a role hierarchy and enabling conditions on roles.

We also established relationships between our spatio-temporal models in terms of expressive power of encoding spatio-temporal requirements. Perhaps, the most important conclusion of our work is that flat RBAC^{+}_{ST} is the most expressive model in the sense that it is able to encode all spatio-temporal requirements that can be expressed in our other models. Therefore, we suggested the use of RBAC^{+}_{ST} to specify spatio-temporal policies when there are small number of users and permissions. On the other hand, when it is necessary to use the role hierarchy, it is rarely helpful to impose spatio-temporal constraints on roles and the role hierarchy; instead, these constraints should be applied to users and user-role assignments.

All existing models, such as GTRBAC, tend to focus on the syntax of temporal and spatio constraints, rather than on the authorization semantics. While we believe that the syntax of temporal and spatial constraints will generally be application-dependent, we did consider the representation of spatio-temporal constraints, distinguishing between concrete and symbolic domains. Concrete domains comprise a set of points defined by some numerically-encoded reference system; symbolic domains comprise sets of labels, each corresponding to one or more points in a concrete domain. In summary, our approach of representing spatio-temporal domain is considerably simpler than existing work, such as GTRBAC.

A number of papers have subsequently appeared on GTRBAC, but this work has ignored temporal constraints and focused on issues related to the multiple role hierarchies [36, 55, 57, 96]. In particular, Du and Joshi [36] define the inter-domain role mapping (IDRM) problem in the context of GTRBAC in which temporal considera-

tions were completely ignored. Moreover, statement of the IDRM problem was not properly defined. In the next chapter, we will investigate this problem and other related problems in a very well understood context, such as RBAC96. We can then easily apply insights or techniques obtained to more complex models, such as ERBAC07 and spatio-temporal RBAC models that are developed in this chapter.

Chapter 5

Set Covering Problems in RBAC

In this chapter we consider the computational complexity of a number of important problems in role-based access control. In fact, it was the analysis of the inter-domain role mapping (IDRM) problem in ERBAC07 and spatio-temporal RBAC models, which leads us to rephrase this problem in a more abstract form that is close to the standard set cover problem. Solving the complexity of this problem enables us to solve the IDRM problem and other similar problems that arise in different types of RBAC models we introduced in previous chapters. We now briefly introduce those RBAC problems.

Du and Joshi [36] defined the IDRM problem, that is to find a set of roles of minimal cardinality such that the authorized permissions for that set of roles is precisely the set of requested permissions. It is easy show that the IDRM problem is not well-defined, because there may not exist a set of roles that are authorized for precisely the set of requested permissions. In this chapter, we provide a more accurate formulation of the IDRM problem from two different perspectives: safety and availability [21]. In terms of availability, we want to find a set of roles that is collectively authorized for the set of requested permissions and also minimizes the number of additional permissions that are granted. Alternatively, from the point of view of safety, we want to find a set of roles

that grant some maximal set of permissions strictly contained in the set of requested permissions. We call the former the *IDRM-availability* problem, and the latter the *IDRM-safety* problem.

Zhang and Joshi [96] then extended the IDRM problem to define the *user-authorization query* (UAQ) problem, which asks whether there exists a set of roles that can be activated in a session for a set of permissions requested by a user. This set of roles must satisfy dynamic separation of duty constraints that impose restrictions on combinations of roles that may be activated (by any user) in a session. In other words, the UAQ problem is an extension of the IDRM problem with constraints on role activation. Wickramaarachchi *et al* [94] considered a more general definition of the UAQ problem by including a lower bound and an upper bound for the requested set of permissions. This general version of the UAQ problem seeks a set of roles such that the requested user can activate that set of roles in a session, and the authorized permissions for that set of roles is between the lower bound and the upper bound of the requested permissions.

On the other hand, Li *et al* [19, 65] recently studied a number of interesting questions regarding separation of duty constraints and their enforcement in the context of role-based access control. A static separation of duty (SSoD) constraint requires that sensitive combinations of permissions should not be available to fewer than k users, but most approaches in the RBAC literature specify such constraints in terms of restrictions on the assignment of users to roles. They mainly showed that SSoD constraints defined on sets of permissions can be enforced by generating constraints on the assignment of users to roles, and gives rise to the following problems.

- Given a RBAC state, and a SSoD constraint, is the SSoD constraint *enforceable* in the RBAC state, in the sense that, does there exist a set of fewer than k

roles that are authorized for the set of sensitive permissions defined in the SSoD constraint?

- Given a RBAC state, and an enforceable SSoD constraint, how to generate a set of RSSoD constraints (that is, constraints defined on the assignment of users to roles) that is equivalent to the SSoD constraint, in the sense that the set of RSSoD constraints is satisfied if and only if the SSoD constraint is satisfied.

We call the former the *enforceability of static separation of duty constraints* problem, and the latter the *generation of role-based static separation of duty (RSSoD) constraints* problem.

However, existing work does not always determine the computational complexity of the problem (instead presenting either heuristic [94, 96] or exhaustive algorithms to compute a solution [65]). All the above problems appear to be related to the well known *set cover problem* [42]: the decision version of this problem is **NP**-complete, while the optimization problem is **NP**-hard. In this chapter, we examine the connections between computational problems arising in RBAC and the set cover problem. Our most important contribution in this chapter is to define the *minimal cover problem* – a generalization of the set cover problem – and use this problem to determine the computational complexity of the IDRM-availability problem and the user-authorization query problem. In doing so, we identify some interesting auxiliary problems and establish their computational complexity. We also establish a vocabulary and a suite of techniques for handling similar problems that may subsequently arise in the context of RBAC. The material in this chapter is mainly derived from previous published papers [21, 23].

This chapter is arranged as follows. In the next section we formally define the set cover problem. Section 5.2 introduces the minimal cover problem and establishes its

relationship to the basic set cover problem, thereby enabling us to derive its computational complexity. In Section 5.3, we discuss applications of our results to RBAC, establishing complexity results for a number of different problems. We also discuss related work in Section 5.3.

5.1 The set cover problem

Let X be a finite set and let $\mathcal{C}$ be a collection of subsets of X such that $X = \bigcup_{C \in \mathcal{C}} C$, and let $\mathcal{D} \subseteq \mathcal{C}$. Then we write $U_{\mathcal{D}}$ to denote $\bigcup_{D \in \mathcal{D}} D$. (By definition, $U_{\mathcal{D}} \subseteq X$ for any $\mathcal{D} \subseteq \mathcal{C}$; in particular, $U_{\mathcal{C}} = X$).

Definition 5.1.1 *Let X be a finite set and let $\mathcal{C}$ be a collection of subsets of X such that $U_{\mathcal{C}} = X$. Let $V \subseteq X$. We say $\mathcal{D} \subseteq \mathcal{C}$ is a* cover *of V if $U_{\mathcal{D}} \supseteq V$; $\mathcal{D}$ is a* perfect cover *of V if $U_{\mathcal{D}} = V$.*

The definition above is more general than the usual definition associated with the set cover problem. In particular, our notion of a "perfect cover" is what usually corresponds to a "cover" in the literature. However, in Section 5.2 we will need to be able to distinguish between covers and perfect covers, hence the more general definition.

Clearly, there exists at least one perfect cover of X (namely $\mathcal{C}$). Note that any cover of X is necessarily perfect, since $U_{\mathcal{C}} = X$. The set cover problem can be expressed in terms of perfect covers.

Problem 5.1.1 (The set cover decision problem) *For a given integer k, does there exist a perfect cover $\mathcal{D}$ of X such that $|\mathcal{D}| \leqslant k$?*

Problem 5.1.2 (The set cover optimization problem) *What is the smallest integer m for which there exists a perfect cover of X of cardinality m?*

The set cover decision problem is **NP**-complete [42] with respect to the parameter $|\mathcal{C}|$. The set cover optimization problem is **NP**-hard, because there exists a (trivial) polynomial time Turing reduction from the set cover decision problem to the set cover optimization problem.[1]

5.2 Variations on the set cover problem

In this section, we introduce some terminology related to the computation of covers and perfect covers. We also define a number of new computational problems using this terminology.

Throughout this section, we assume we are given a universe X and $\mathcal{C}$, a collection of subsets of X. We define a binary relation $\sim$ on the powerset of $\mathcal{C}$: $\mathcal{D} \sim \mathcal{D}'$ if and only if $U_{\mathcal{D}} = U_{\mathcal{D}'}$.

Proposition 5.2.1 *$\sim$ is an equivalence relation.*

Proof Given $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3 \subseteq \mathcal{C}$, we show that $\sim$ is reflexive, symmetric and transitive. Clearly, $\sim$ is reflexive, that is $\mathcal{D}_1 \sim \mathcal{D}_1$, since $U_{\mathcal{D}_1} = U_{\mathcal{D}_1}$. If $U_{\mathcal{D}_1} = U_{\mathcal{D}_2}$, then $U_{\mathcal{D}_2} = U_{\mathcal{D}_1}$, which implies that if $\mathcal{D}_1 \sim \mathcal{D}_2$, then $\mathcal{D}_2 \sim \mathcal{D}_1$. Hence $\sim$ is symmetric. Similarly, if $U_{\mathcal{D}_1} = U_{\mathcal{D}_2}$ and $U_{\mathcal{D}_2} = U_{\mathcal{D}_3}$, then $U_{\mathcal{D}_1} = U_{\mathcal{D}_3}$. Hence $\sim$ is also transitive.■

The equivalence classes defined by $\sim$ give rise to a partition of the powerset of $\mathcal{C}$: the elements of an equivalence class are all subsets of $\mathcal{C}$, and all elements in an equivalence class are perfect covers of the same subset of X. If there exists a perfect cover of $V \subseteq X$ – that is, there exists $\mathcal{D} \subseteq \mathcal{C}$ such that $U_{\mathcal{D}} = V$ – then we write $[V] \subseteq \mathcal{C}$ to denote the equivalence class in which each element of $[V]$ is a perfect cover of V. That is, $[V] = \{\mathcal{D} \subseteq \mathcal{C} : U_{\mathcal{D}} = V\}$.

[1] If we have an oracle that can solve the optimization problem, we can solve the decision problem by checking whether the solution of the associated optimization problem has cardinality less than or equal to k.

We write $\mathsf{PCov}(X, \mathcal{C})$ to denote the set of subsets of X for which perfect covers exist in $\mathcal{C}$. Clearly, $(\mathsf{PCov}(X, \mathcal{C}), \subseteq)$ is a partially ordered set. When X and $\mathcal{C}$ are obvious from context, we will simply write PCov for $\mathsf{PCov}(X, \mathcal{C})$.

Example 5.2.1 *Let* $X = \{1, 2, 3, 4\}$ *and let* $\mathcal{C} = \{C_1, C_2, C_3, C_4\}$, *where* $C_1 = \{1\}$, $C_2 = \{2, 4\}$, $C_3 = \{3, 4\}$ *and* $C_4 = \{1, 2, 4\}$. *Then*

$$\mathsf{PCov} = \{\{1\}, \{2, 4\}, \{3, 4\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}, \{1, 2, 3, 4\}\}$$

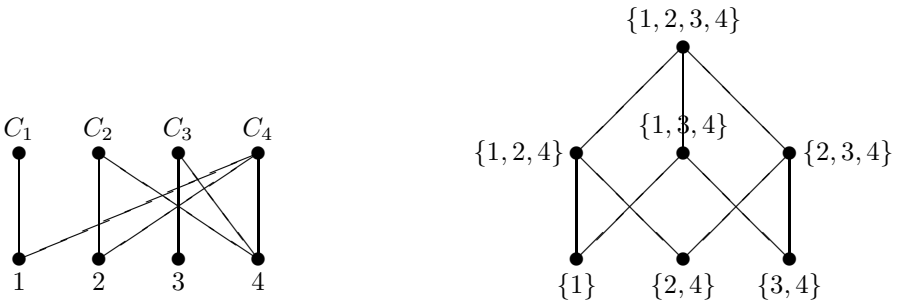
and

$$\begin{aligned}
[\{1\}] &= \{\{C_1\}\} \\
[\{2, 4\}] &= \{\{C_2\}\} \\
[\{3, 4\}] &= \{\{C_3\}\} \\
[\{1, 2, 4\}] &= \{\{C_4\}, \{C_1, C_4\}, \{C_2, C_4\}, \{C_1, C_2\}, \{C_1, C_2, C_4\}\} \\
[\{1, 3, 4\}] &= \{\{C_1, C_3\}\} \\
[\{2, 3, 4\}] &= \{\{C_2, C_3\}\} \\
[\{1, 2, 3, 4\}] &= \{\{C_3, C_4\}, \{C_1, C_2, C_3\}, \{C_1, C_3, C_4\}, \{C_2, C_3, C_4\}, \{C_1, C_2, C_3, C_4\}\}
\end{aligned}$$

Figure 5.1 shows a graphical representation of X *and* $\mathcal{C}$, *and a Hasse diagram of the partially ordered set* $(\mathsf{PCov}, \subseteq)$. *Clearly,* $\{C_3, C_4\}$ *is a solution to this instance of the set cover optimization problem.*

5.2.1 The kernel and shell

We now introduce the notion of the kernel and shell of V (given X and $\mathcal{C}$). Informally, the kernel of V represents the largest subset of X that is perfectly covered and is



(a) $\mathcal{C}$ and X (b) PCov

Figure 5.1: A graphical representation of X and $\mathcal{C}$, and a Hasse diagram of PCov

contained in V. We shall see that the kernel of V can be computed in polynomial time, a result that has a number of useful applications. The shell identifies those sets that could contribute to a cover of V.

Definition 5.2.1 *Let $V \subseteq X$. Define $\mathcal{K}(V) = \{C \in \mathcal{C} : C \subseteq V\}$. Then we call $U_{\mathcal{K}(V)} \subseteq X$ the* kernel *of V (with respect to $\mathcal{C}$).*

For brevity, we write $\mathsf{ker}(V)$, rather than $U_{\mathcal{K}(V)}$, to denote the kernel of V. Note that $\mathsf{ker}(V) \in \mathsf{PCov}$ and $\mathsf{ker}(V) \subseteq V$, by definition. We now state and prove two elementary results.

Proposition 5.2.2 *Let $Z \in \mathsf{PCov}$ such that $Z \subseteq V$. Then $Z \subseteq \mathsf{ker}(V)$.*

Proof Since $Z \in \mathsf{PCov}$, there exists $\mathcal{D} \subseteq \mathcal{C}$ such that $Z = U_{\mathcal{D}}$. For any $C \in \mathcal{D}$, we have $C \subseteq V$ (otherwise, $Z \not\subseteq V$). Hence, $C \in \mathcal{K}(V)$ by definition and hence $\mathcal{D} \subseteq \mathcal{K}(V)$. Therefore $Z = U_{\mathcal{D}} \subseteq U_{\mathcal{K}(V)} = \mathsf{ker}(V)$. ■

Proposition 5.2.3 *$V \in \mathsf{PCov}$ if and only if $V = \mathsf{ker}(V)$.*

Proof The result follows immediately if $V = \mathsf{ker}(V)$ since $\mathsf{ker}(V) \in \mathsf{PCov}$. Assume now that $V \in \mathsf{PCov}$. Since $V \subseteq V$, we may apply Proposition 5.2.2 to deduce that $V \subseteq \mathsf{ker}(V)$. Hence, we have $V = \mathsf{ker}(V)$, since $\mathsf{ker}(V) \subseteq V$, by definition. ■

Corollary 5.2.1 *Let $V \subseteq X$. Determining whether $V \in \mathsf{PCov}$ is in $\boldsymbol{P}$.*

Proof By Proposition 5.2.3, $V \in \mathsf{PCov}$ if and only if $V = \mathsf{ker}(V)$. Clearly, we can check whether $V = \mathsf{ker}(V)$ in polynomial time with respect to $|\mathcal{C}|$ and $|V|$. ■

Definition 5.2.2 *Let $V \subseteq X$. Define $\mathcal{S}(V) = \{C \in \mathcal{C} : C \cap V \neq \emptyset\}$. Then we call $U_{\mathcal{S}(V)} \subseteq X$ the* shell *of V (with respect to $\mathcal{C}$).*

Similarly, we write $\mathsf{shell}(V)$ to denote the shell of V. Note that $\mathsf{shell}(V) \in \mathsf{PCov}$ and $\mathsf{shell}(V) \supseteq V$, by definition.

5.2.2 Minimality, optimality and irreducibility

Let us assume that $V \notin \mathsf{PCov}$ and consider the problem of finding an "approximation" of V among the members of PCov. (We will formalize the notion of approximation shortly.) The results above suggest that the best "under-approximation" of V is $\mathsf{ker}(V)$. It seems natural to consider "over-approximation" in terms of those elements of PCov that contain V and have minimal cardinality. More formally, we have the following definitions.

Definition 5.2.3 *Given X, $\mathcal{C}$ and $V \subseteq X$ such that $V \notin \mathsf{PCov}$, we say:*

- *$T \in \mathsf{PCov}$ is a* container *of V if $T \supset V$;*
- *$T \in \mathsf{PCov}$ is a* minimal container *of V if T is a container of V and for any other container T' of V, $|T| \leqslant |T'|$.*[2]

In other words, T is a minimal container of V if it is perfectly covered by some subset of $\mathcal{C}$, contains V, but contains as few elements outside V as possible for a set that is perfectly covered.[3]

[2]Equivalently, there does not exist $T' \in \mathsf{PCov}$ such that $T' \supseteq V$ and $|T'| < |T|$.

[3]This is important in the context of RBAC because we want to minimize the number of additional permissions for which a set of roles is authorized outside some specified set of permissions.

Definition 5.2.4 *Given X, $\mathcal{C}$ and $V \subseteq X$ such that $V \notin \mathsf{PCov}$, we say:*

- $\mathcal{D} \subseteq \mathcal{C}$ *is* irreducible *if for all $\mathcal{D}' \subset \mathcal{D}$, $U_{\mathcal{D}'} \subset U_{\mathcal{D}}$;*
- $\mathcal{D} \subseteq \mathcal{C}$ *is a* minimal cover *of V if $\mathcal{D} \in [T]$ for some minimal container T of V;*
- $\mathcal{D} \in [T]$ *is an* optimal cover *of V if T is a minimal container of V and $\mathcal{D}$ is irreducible.*

Informally, $\mathcal{D}$ is irreducible if there is no redundancy in $\mathcal{D}$: we cannot remove any element of $\mathcal{D}$ without changing $U_{\mathcal{D}}$. Each $T \in \mathsf{PCov}$ is associated with the equivalence class $[T]$, which is a collection of subsets of $\mathcal{C}$, and every member of $[T]$ is a perfect cover of T. If T is a minimal container of V, then every element of $[T]$ is a minimal cover of V. Each such equivalence class contains at least one irreducible element.

Example 5.2.2 *Using our running example, let $V = \{1, 2, 3\}$. Then a minimal container of $\{1, 2, 3\}$ is $\{1, 2, 3, 4\}$. The irreducible covers in $[\{1, 2, 3, 4\}]$ (and hence optimal covers of $\{1, 2, 3\}$) are $\{C_3, C_4\}$ and $\{C_1, C_2, C_3\}$.*

Proposition 5.2.4 *Given $\mathcal{D} \subseteq \mathcal{C}$, we can compute $\mathcal{E} \subseteq \mathcal{D}$ such that $\mathcal{E}$ is irreducible and $U_{\mathcal{E}} = U_{\mathcal{D}}$ in polynomial time.*

Proof Figure 5.2 illustrates an algorithm called IRR-Gen: on input $\mathcal{D} \subseteq \mathcal{C}$, IRR-Gen returns an irreducible set $\mathcal{E} \subseteq \mathcal{D}$ such that $U_{\mathcal{E}} = U_{\mathcal{D}}$. At the ith iteration, the algorithm arbitrarily chooses an element C from $\mathcal{D}$, and checks whether the removal of C from $\mathcal{D}$ would affect the set of elements originally covered by $\mathcal{D}$. If it does, C must be included in $\mathcal{E}$, otherwise C can be ignored. The overall time complexity of the IRR-Gen algorithm is polynomial in $|\mathcal{D}|$ and $|X|$, that is, there are $|\mathcal{D}| \leqslant |\mathcal{C}|$ iterations of the while loop, and the subset inclusion test can be implemented as a loop of no more than $|X|$ iterations. ■

```
Input: 𝒟 ⊆ 𝒞
Output: ℰ
let ℰ = ∅;
while 𝒟 ≠ ∅ {
  choose C ∈ 𝒟;
  𝒟 = 𝒟 \ {C};
  if C ⊈ U_{𝒟∪ℰ}
    then ℰ = ℰ ∪ {C};
}
return ℰ
```

Figure 5.2: The IRR-Gen algorithm

Note that IRR-Gen is non-deterministic ("choose $C \in \mathcal{D}$") and $[T]$ may contain more than one irreducible set, so different runs of the algorithm on input $\mathcal{D} \in [T]$ might return different irreducible sets $\mathcal{E} \in [T]$ depending on the order in which the elements of $\mathcal{D}$ are processed.

Example 5.2.3 *Using our running example, let* $\mathcal{D} = \{C_1, C_2, C_3, C_4\} \in [\{1, 2, 3, 4\}]$. *Then processing* $\mathcal{D}$ *in the order* C_1, C_2, C_3, C_4, *for example, yields* $\mathcal{E} = \{C_3, C_4\}$, *whereas processing* $\mathcal{D}$ *in the order* C_4, C_3, C_2, C_1 *yields* $\mathcal{E} = \{C_1, C_2, C_3\}$.

Corollary 5.2.2 *Given* X, $\mathcal{C}$ *and* $T \in \mathsf{PCov}$, *we can compute an irreducible element of* $[T]$ *in polynomial time.*

Proof Since $T \in \mathsf{PCov}$, $T = \mathsf{ker}(T)$ by Proposition 5.2.3. $\mathcal{K}(T) = \{C \in \mathcal{C} : C \subseteq T\}$, we can always compute $\mathcal{K}(T)$ in polynomial time. Moreover, $U_{\mathcal{K}(T)} = \mathsf{ker}(T) = T$, hence $\mathcal{K}(T) \in [T]$. Then we can compute an irreducible element of $[T]$ using the IRR-Gen algorithm with input $\mathcal{K}(T)$. ■

5.2.3 The minimal cover problem

We can now define the minimal cover problem, and address its computational complexity. This, in turn will allow us to address the IDRM-availability problem.

Let $V \notin \mathsf{PCov}$ and suppose we are interested in finding a minimal cover of V. We first state two simplifying assumptions on the problem instance that enable us to

eliminate "inessential" aspects for finding a minimal cover of V. In other words, we construct (in polynomial time) a new instance of the problem, by replacing X and $\mathcal{C}$ with X' and $\mathcal{C}'$, where $|X| \geqslant |X'|$ and $|\mathcal{C}| \geqslant |\mathcal{C}'|$. In particular, we omit any $C \in \mathcal{C}$ such that

- $C \cap V = \emptyset$ (since any such C cannot contribute to a cover of V);
- $C \subseteq V$ (since, by Proposition 5.2.5, we can compute a minimal cover $\mathcal{D}$ of $V \setminus \mathsf{ker}(V)$ to obtain a minimal cover $\mathcal{D} \cup \mathcal{K}(V)$ of V).

Proposition 5.2.5 *Given X, V and $\mathcal{C}$, define: $X' = X \setminus \mathsf{ker}(V)$; $V' = V \setminus \mathsf{ker}(V)$; and $\mathcal{C}' = \{C \setminus \mathsf{ker}(V) : C \in \mathcal{C}, C \not\subseteq V\}$. Then:*

1. *$U_{\mathcal{C}'} = X'$;*
2. *for all $C' \in \mathcal{C}'$, $C' \not\subseteq V'$;*
3. *if $\mathcal{D}$ is a minimal cover of V', then $\mathcal{D} \cup \mathcal{K}(V)$ is a minimal cover of V.*

Proof

1. Since $\mathcal{C}' = \{C \setminus \mathsf{ker}(V) : C \in \mathcal{C}, C \not\subseteq V\}$, $U_{\mathcal{C}'} = U_{\mathcal{C}} \setminus \mathsf{ker}(V) = X \setminus \mathsf{ker}(V) = X'$.
2. If $C' \in \mathcal{C}'$, then $C' = C \setminus \mathsf{ker}(V)$ for some $C \in \mathcal{C}$ such that $C \not\subseteq V$. Therefore, $C' = C \setminus \mathsf{ker}(V) \not\subseteq V \setminus \mathsf{ker}(V) = V'$.
3. Suppose, in order to obtain a contradiction, that $\mathcal{D} \cup \mathcal{K}(V)$ is not a minimal cover of V. Then, since $\mathcal{K}(V)$ only adds elements from V, $\mathcal{D}$ cannot be a minimal cover of V', which is the desired contradiction.

■

Henceforth, we assume that our problem instance is in this "canonical form": that is, $C \cap V \neq \emptyset$ and $C \not\subseteq V$ for all $C \in \mathcal{C}$. We now define a number of problems associated with containers, minimal covers and optimal covers.

Problem 5.2.1 (The container decision problem) *Given X, $\mathcal{C}$, $V \subseteq X$ and an integer k, does there exist a container T of V such that $|T| \leqslant |V| + k$?*

Problem 5.2.2 (The container optimization problem) *Given X, $\mathcal{C}$ and $V \subseteq X$, find a minimal container of V.*

Problem 5.2.3 (The minimal cover problem) *Given X, $\mathcal{C}$ and $V \subseteq X$, find a minimal cover of V.*

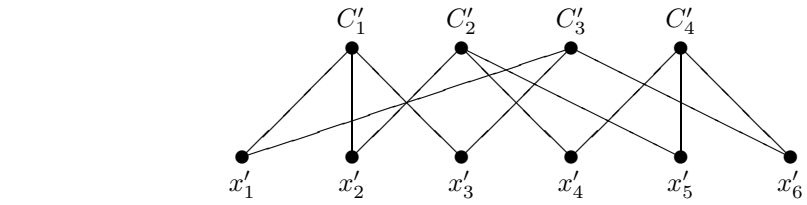
Problem 5.2.4 (The optimal cover problem) *Given X, $\mathcal{C}$ and $V \subseteq X$, find an optimal cover of V.*

Theorem 5.2.1 *The container decision problem is **NP**-complete.*

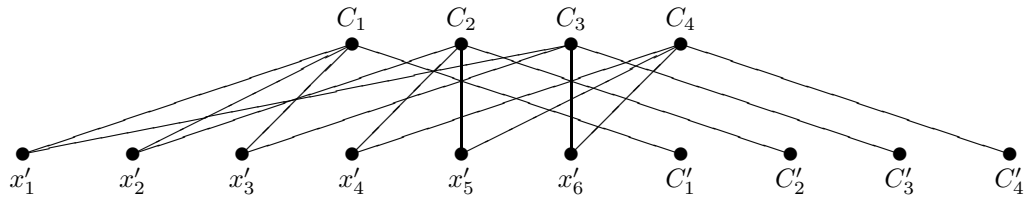
Proof It is easy to see that the container decision problem is in **NP**, because a nondeterministic algorithm need only guess a subset T of X and check in polynomial time whether $T \supset V$, $\mathsf{ker}(T) = T$ (that is, $T \in \mathsf{PCov}$) and $|T| \leqslant |V| + k$.

We now show a polynomial time transformation from the set cover decision problem to a special case of the container decision problem. Let $(X', \mathcal{C}', k)$ be an instance of the set cover decision problem. We transform it into an instance $(X, \mathcal{C}, V, k)$ of a special case of the container decision problem in the following way: define $X = X' \cup \mathcal{C}'$, $V = X'$, and $\mathcal{C} = \{C' \cup \{C'\} : C' \in \mathcal{C}'\}$. This transformation is illustrated in Figure 5.3. It can be seen that each C_i contains a single element (namely C'_i) that does not belong to V. Moreover, each C_i contains at least one element of X', since $C'_i \in \mathcal{C}'$ can be assumed to be non-empty. In other words, the resulting instance is a special case of the container decision problem (in which each element of $\mathcal{C}$ contains precisely one distinct element that is not in V).

We now show that there exists a set cover $\mathcal{D}' \subseteq \mathcal{C}'$ of size k if and only if there exists a container $U_{\mathcal{D}}$ of V such that $|U_{\mathcal{D}}| = |V| + k$. First, suppose $\mathcal{D}' = \{C'_1, \ldots, C'_k\}$



(a) A set cover problem



(b) A special case of the container problem

Figure 5.3: Correspondence between the set cover and container problems

is a set cover (of X'). Then, by construction, $\mathcal{D} = \{C_1, \ldots, C_k\}$ is a cover of $V = X'$ and $|U_{\mathcal{D}}| = |V| + k$, as required.

Conversely, suppose there exists a container $U_{\mathcal{D}}$ of V with size $|U_{\mathcal{D}}| = |V| + k$. Then, by construction, $\mathcal{D} = \{C_1, \ldots, C_k\}$ is a cover of $V = X'$. Again, by construction, $\mathcal{D}' = \{C'_1, \ldots, C'_k\}$ is a set cover of X'. ■

Corollary 5.2.3 *The container optimization problem is **NP**-hard.*

Proof The result follows from the fact that the associated decision problem is **NP**-complete (or we can use the construction illustrated in Figure 5.3 to solve the set cover optimization problem using the container optimization problem). ■

Corollary 5.2.4 *The minimal cover problem is **NP**-hard.*

Proof We exhibit a polynomial time Turing reduction from the container optimization problem to the minimal cover problem. Suppose there exists an oracle for the minimal cover problem. Then given an instance $(X, \mathcal{C}, V)$ of the container optimization problem, we query the oracle for the minimal cover problem on instance $(X, \mathcal{C}, V)$, to obtain a minimal cover $\mathcal{D} \subseteq \mathcal{C}$ of V. Then we simply compute $U_{\mathcal{D}} \subseteq X$, which is, by definition, a minimal container of V. ■

Corollary 5.2.5 *The optimal cover problem is **NP**-hard.*

Proof We show that the minimal cover problem is polynomial time Turing equivalent to the optimal cover problem. Clearly, any solution for the optimal cover problem is a solution for the minimal cover problem. We now show a polynomial time Turing reduction from the optimal cover problem to the minimal cover problem. Given any instance $(X, \mathcal{C}, V)$ of the optimal cover problem, we query an oracle to obtain a solution $\mathcal{D}$ for the minimal cover problem. We can then compute $\mathcal{D}' = \text{IRR-Gen}(\mathcal{D})$ in polynomial time, which is a solution to the optimal cover problem. ■

5.2.4 The irreducible cover problem

In this section, we will not be concerned with containers of V. Instead we will be concerned with all covers of X that are irreducible. We say $\mathcal{D}$ is an *irreducible cover* of X if $\mathcal{D}$ is irreducible and $U_{\mathcal{D}} = X$.

Problem 5.2.5 (The irreducible cover decision problem) *Given X, $\mathcal{C}$ and a positive integer k, does there exist $\mathcal{D} \subseteq \mathcal{C}$ such that $\mathcal{D}$ is an irreducible cover of X and $|\mathcal{D}| \leqslant k$?*

Problem 5.2.6 (The irreducible cover optimization problem) *Given X and $\mathcal{C}$, find $\mathcal{D} \subseteq \mathcal{C}$ such that $\mathcal{D}$ is an irreducible cover of X and $|\mathcal{D}|$ is minimized.*

Problem 5.2.7 (The irreducible cover enumeration problem) *Given X and $\mathcal{C}$, find all $\mathcal{D} \subseteq \mathcal{C}$ such that $\mathcal{D}$ is an irreducible cover of X.*

Theorem 5.2.2 *The irreducible cover decision problem is **NP**-complete. The irreducible cover optimization and enumeration problems are **NP**-hard.*

Proof It is easy to see that the irreducible cover decision problem is in **NP**, because a nondeterministic algorithm need only guess a subset $\mathcal{D}$ of $\mathcal{C}$ and check whether $\mathcal{D}$ is an

irreducible cover of X and $|\mathcal{D}| \leqslant k$. Checking whether $\mathcal{D}$ is an irreducible cover of X can be done in polynomial time by checking whether $U_{\mathcal{D}} = X$ and checking whether $\mathcal{D}$ is irreducible can be done in polynomial time by confirming whether $\mathcal{D} = \text{IRR-Gen}(\mathcal{D})$.

Clearly, we can use an algorithm that solves the irreducible cover problem to solve the set cover problem. It is obvious that there is an irreducible cover of cardinality less than or equal to k if and only if there is some cover of cardinality less than or equal to k.

There are trivial polynomial time Turing reductions from the irreducible cover decision problem to both the irreducible cover optimization and irreducible cover enumeration problems. In the first case, we query an oracle for the optimization problem and return "yes" for the decision problem if the cardinality of the cover returned by the oracle is less than or equal to k. In the second case, let us assume that the oracle returns a list of irreducible covers, and this list of irreducible covers can be sorted in order of increasing cardinality in polynomial time. Then to solve the decision problem, we simply need to determine whether the cardinality of the first element in the list is less than or equal to k. ■

5.3 Covering problems in RBAC

The results of the previous section, particularly those on problems associated with minimal containers, may be of independent mathematical interest, but in this section we apply these results to a number of problems in the RBAC literature.

In this section we will assume that the role hierarchy has been "flattened" by encoding all authorized relationships in the user-role and permission-role relations, so that RBAC state is simply represented by the RBAC_0 state (UA, PA). Any RBAC_1 state can be transformed into an equivalent RBAC_0 state (in the sense that precisely the

same set of requests are authorized) in polynomial time, using an algorithm based on some appropriate graph traversal algorithm.

Given an instance (R, P, PA) of the RBAC_0 model and an instance $(X, \mathcal{C})$ of the set cover problem, P is synonymous with X and $\{\mathsf{Prms}(r) : r \in R\}$[4] is synonymous with $\mathcal{C}$. (This assumes that each role is assigned to at least one permission in P, and each permission is assigned to at least one role in R.)

Given $Q \subseteq P$, $\mathcal{K}(Q)$ comprises those sets of permissions that are contained within Q. In other words, $\mathcal{K}(Q)$ is synonymous with those roles that are only authorized for permissions in Q. Similarly, $\mathcal{S}(Q)$ is synonymous with those roles that are authorized for at least one permission in Q.

5.3.1 The inter-domain role mapping problem

In loosely-coupled distributed environments, users' identities are usually not known in advance to resource owners. Piromruen and Joshi [74] propose a requirement-driven interoperation approach that maps requests from users in an external domain to RBAC policies in the target domain. Hence, the underlying problem is to find a set of hierarchically related roles in the target domain that are authorized for the requested set of permissions. In order to observe the principle of least privilege, we want to find a set of roles such that the set of permissions acquired by activating that set of roles *approximates* the set of requested permissions as closely as possible. Du and Joshi [36] refer to this problem as the *inter-domain role mapping* (IDRM) problem. Their statement of the IDRM problem is to find a set of roles of minimal cardinality such that the authorized permissions for that set of roles is precisely the set of requested permissions. We formally define the problem as follows.

[4]Recall that $\mathsf{Prms}(r) = \{p \in P : (p, r) \in PA\}$.

Problem 5.3.1 (The IDRM problem) *Given R, P, $PA \subseteq P \times R$ and $Q \subseteq P$, find $S \subseteq R$ such that $\mathsf{Prms}(S) = Q$ and $|S|$ is minimized.*

It is worth noting that many instances of the IDRM problem, as defined above, may not have a solution, since there may not exist $S \subseteq R$ such that $\mathsf{Prms}(S) = Q$. Hence, we define a preliminary question.

Problem 5.3.2 (The preliminary IDRM problem) *Given R, P, PA and $Q \subseteq P$, does there exist $R_Q \subseteq R$ such that $\mathsf{Prms}(R_Q) = Q$?*

We first note that Problem 5.3.2 can be decided in polynomial time, since it can be answered by determining whether $Q = \mathsf{ker}(Q)$. If so, then $R_Q = \mathcal{K}(Q)$. Having answered the preliminary IDRM problem, we may then pose the following problems.

Problem 5.3.3 (The exact IDRM decision problem) *Given R, P, PA, $Q \subseteq P$, $R_Q \subseteq R$ such that $\mathsf{Prms}(R_Q) = Q$, and an integer k, does there exist $S \subseteq R_Q$ such that $\mathsf{Prms}(S) = Q$ and $|S| \leqslant k$.*

Problem 5.3.4 (The exact IDRM optimization problem) *Given R, P, PA, $Q \subseteq P$, and $R_Q \subseteq R$ such that $\mathsf{Prms}(R_Q) = Q$, find $S \subseteq R_Q$ such that $\mathsf{Prms}(S) = Q$ and $|S|$ is minimized.*

Clearly, the set cover decision problem is identical to the exact IDRM decision problem. Given any instance $(X, \mathcal{C}, k)$ of the set cover decision problem, we simply set $Q = X$, and for each $C \in \mathcal{C}$, define $r_C \in R_Q$ and $\mathsf{Prms}(r_C) = C$. Then k members of R_Q cover Q if and only if k members of $\mathcal{C}$ cover X. In other words, the exact IDRM decision problem is **NP**-complete, and the exact IDRM optimization problem is **NP**-hard.

It is also worth observing that there appears to be no good reason to minimize $|S|$ (in the statement of the IDRM problem): it is not clear why S is preferable to

S' if $\mathsf{Prms}(S) = \mathsf{Prms}(S')$ and $|S| < |S'|$. Moreover, if there is no solution to the IDRM problem (that is, there does not exist $S \subseteq R$ such that $\mathsf{Prms}(S) = Q$) then it is the permissions for which an approximate solution S is authorized that should be of interest, rather than $|S|$. In order to address concerns about the appropriateness of the IDRM problem, we introduced two problems derived from the IDRM problem [21].

Problem 5.3.5 (The IDRM-safety problem) *Given P, R, PA and $Q \subseteq P$, find $S \subseteq R$ such that $\mathsf{Prms}(S) \subseteq Q$ and $|\mathsf{Prms}(S)|$ is maximized.*

Problem 5.3.6 (The IDRM-availability problem) *Given P, R, PA and $Q \subseteq P$, find $S \subseteq R$ such that $\mathsf{Prms}(S) \supseteq Q$ and $|\mathsf{Prms}(S)|$ is minimized.*

The IDRM-safety problem is concerned with ensuring that no permission outside Q is authorized for any role in S, while authorizing S for as many permissions as possible in Q. The availability approach to IDRM ensures that all permissions in Q are authorized for at least one role in S, but seeks to minimize the number of additional permissions for which S is authorized.

Although there is an obvious correspondence between the exact IDRM problem and the set cover problem (as we illustrated above), there is no obvious way of transforming the IDRM-availability problem to the set cover problem, since we are simultaneously concerned with covering Q while minimizing what is covered outside Q. Clearly, however, the IDRM-availability problem does map very easily to, and is no harder than, the minimal cover problem discussed in Section 5.2. More formally, we have the following result.

Theorem 5.3.1 *The IDRM-safety problem is in **P**; the IDRM-availability problem is **NP**-hard.*

Proof The largest subset of Q for which a perfect cover exists is, by Proposition 5.2.2,

$\mathsf{ker}(Q)$ which can be computed in polynomial time. Hence, the IDRM-safety problem is in **P**.

We now exhibit a polynomial time Turing reduction from the minimal cover problem to the IDRM-availability problem. Given any instance $(X, \mathcal{C}, V)$ of the minimal cover problem, we can transform it into an instance (P, Q, R, PA) of the IDRM-availability problem in polynomial time. In particular, we let $X = P$, $V = Q$, and for each $C \in \mathcal{C}$, define $r_C \in R$ and $\mathsf{Prms}(r_C) = C \subseteq X = P$. Clearly, a solution $S \subseteq R$ to this instance of the IDRM-availability problem provides a solution to the given instance of the minimal cover problem. ■

5.3.2 The user authorization query problem

In an open and distributed system, service providers usually do not know identities of service consumers or clients in advance, and predefined roles are assigned to users based on the attributes one or more trust authorities assert they possess. Similarly, clients usually do not know which roles are assigned to them, and instead directly request to access resources from service providers. Hence, it is the job of service providers to check whether clients are authorized to perform all their requested permissions, and correspondingly return a set of roles for clients to activate, and thereby acquire the appropriate set of permissions for which they are authorized.

Zhang and Joshi [96] recently defined the *user authorization query* (UAQ) problem: that is, given a set of permissions $Q \subseteq P$ and a user $u \in U$, does there exist a set of roles $R_Q \subseteq R$ such that u can activate all roles in R_Q and $\mathsf{Prms}(R_Q)$ is "close" to Q. Their solution to the UAQ problem is to first use a greedy algorithm to compute a set of roles R_Q such that $\mathsf{Prms}(R_Q)$ is the best approximation of Q, and then check whether u is allowed to activate all roles in R_Q with regard to constraints defined on role activation. Clearly, the purpose of the first step is to solve the IDRM-safety and IDRM-availability

problems. In other words, if we are concerned with under-approximation of Q, we can find $R_Q \in R$ in polynomial time by computing $R_Q = \mathcal{K}(Q)$. On the other hand, it is difficult to find $R_Q \in R$ such that $\mathsf{Prms}(R_Q)$ is the best over-approximation of Q (since the IDRM-availability problem is **NP**-hard).

However, there are at least two problems with their approach to solving the UAQ problem. Since the IDRM-availability problem is **NP**-hard, there is no guarantee that their greedy algorithm will produce a most appropriate solution R_Q. In addition, it completely ignores the fact that u may not be able to activate one or more of roles in R_Q which is obtained in the first step. In other words, a lot of effort is expended in computing an approximate R_Q (solving IDRM-availability problem) that may not be of any relevance to the user anyway.

Wickramaarachchi *et al* [94] provided a more general definition of the UAQ problem, which is formally stated below.

Problem 5.3.7 (The UAQ problem) *Given P, R, PA and (P_l, P_u, obj), where $P_l, P_u \subseteq P$ and $obj \in \{\max, \min\}$, find $S \subseteq R$ such that the following conditions hold:*

- $P_l \subseteq \mathsf{Prms}(S) \subseteq P_u$ *and* $|\mathsf{Prms}(S)|$ *is maximized if* $obj = \max$*;*
- $P_l \subseteq \mathsf{Prms}(S) \subseteq P_u$ *and* $|\mathsf{Prms}(S)|$ *is minimized if* $obj = \min$.[5]

Let us consider the problem of finding $Q \subseteq P$ such that Q is perfectly covered and $P_l \subseteq Q \subseteq P_u$. Then we can find $S \subseteq R$ that solves the UAQ problem in polynomial time by computing $S = \mathcal{K}(Q)$, since $\mathsf{ker}(Q) = Q$.

[5]In the original paper [94], given a set of constraints C and a user u, they require that u can activate the set of roles S without violating any constraint in C. There is also an additional condition on the cardinality of the solution set S (which essentially requires the computation of either a maximal or minimal element in the appropriate equivalence class). We omit these considerations, which do not affect the complexity of the problem, for clarity and simplicity.

First note that we can compute $\mathsf{ker}(P_u)$ in polynomial time. Note also that for any solution Q, we must have $Q \subseteq \mathsf{ker}(P_u)$, by Proposition 5.2.2, since Q is perfectly covered and $Q \subseteq P_u$. Then three cases must be considered:

1. $P_l \subseteq \mathsf{ker}(P_u)$ and $obj = \max$;

2. $P_l \subseteq \mathsf{ker}(P_u)$ and $obj = \min$;

3. $P_l \not\subseteq \mathsf{ker}(P_u)$.

Case (3) means that no such Q can be found, since $Q \subseteq \mathsf{ker}(P_u)$. In other words, the UAQ problem posed by Wickramaarachchi *et al* only has a solution if $P_l \subseteq \mathsf{ker}(P_u)$. For case (1), we can simply take $Q = \mathsf{ker}(P_u)$, by Proposition 5.2.2. Hence, the only form of the problem that cannot be answered in polynomial time is $(P_l, P_u, \min)$. Henceforth, we restrict our attention to UAQ problems of this form.

Theorem 5.3.2 *The UAQ problem and the container optimization problem are polynomial time Turing equivalent.*

Proof We first show that there is a polynomial time Turing reduction from UAQ to container optimzation. We have to find the smallest Q such that Q is perfectly covered and $P_l \subseteq Q \subseteq \mathsf{ker}(P_u)$. We define $R_{\text{new}} = \{r \in R : \mathsf{Prms}(r) \subseteq P_u\}$ and $P_{\text{new}} = \mathsf{ker}(P_u)$. Then to answer the UAQ instance, we need only answer the container optimization instance for $X = P_{\text{new}}$, $V = P_l$ and $\mathcal{C} = \{\mathsf{Prms}(r) : r \in R_{\text{new}}\}$.

To complete the proof, we show that there is a polynomial time Turing reduction from container optimization to UAQ. The obvious transformation, previously used in the proof of Theorem 5.3.1, suffices. ■

5.3.3 Separation of duty

The work of Li *et al* has been important in identifying and clarifying a number of issues associated with the enforcement of separation of duty constraints in the context of RBAC [19, 65]. We define some notations associated with separation of duty and their precise meanings, which have not been formally defined in the literature [65].

Notation 5.3.1 *Given a set of permissions $Q \subseteq P$ and an integer k such that $1 < k \leqslant |Q|$, $\mathsf{ssod}(Q, k)$ is a* static separation of duty constraint.

Definition 5.3.1 *The constraint $\mathsf{ssod}(Q, k)$ is violated by the RBAC state (PA, UA, RH) if there exists a set of $k - 1$ (or fewer) users that are authorized collectively for all permissions in Q.*

Notation 5.3.2 *Given a set of roles $R' \subseteq R$ and an integer k such that $1 < k \leqslant |R'|$, $\mathsf{rssod}(R', k)$ is a* role-based static separation of duty constraint.

Definition 5.3.2 *The constraint $\mathsf{rssod}(R', k)$ is violated by the RBAC state (PA, UA, RH) if there exists a set of $k - 1$ (or fewer) users that cover R' in the sense that every role is assigned to at least one of those users.*

Notation 5.3.3 *Given a set of roles $R' \subseteq R$ and an integer k such that $1 < k \leqslant |R'|$, $\mathsf{smer}(R', k)$ is a* static mutually exclusive role constraint.

Definition 5.3.3 *The constraint $\mathsf{smer}(R', k)$ is violated by the RBAC state (PA, UA, RH) if there exists a user that is assigned to k or more roles in R'.*

Li *et al* were concerned with re-writing an SSoD constraint in terms of SMER constraints, in such a way that the satisfaction of the SMER constraints implied the satisfaction of the SSoD constraint. A SSoD constraint is said to be *enforceable* if and only if there exists a set of SMER constraints such that the satisfaction of those SMER

constraints implies the satisfaction of the SSoD constraint. Clearly, a SSoD constraint $\mathsf{ssod}(Q, k)$ is not enforceable if there exists a set of $k-1$ roles that are collectively authorized for all the permissions in Q, because we can assign a different user to each of the $k-1$ roles without violating any SMER constraint we can specify, but results in violation of the SSoD constraint. Hence, it is of interest to know whether the SSoD constraint is enforceable using SMER constraints. We now describe three problems associated with the enforcement of separation of duty constraints.

Problem 5.3.8 (The SSoD enforceability decision problem) *Given P, R, PA, $Q \subseteq P$ and an integer k, does there exist $S \subseteq R$ such that $\mathsf{Prms}(S) \supseteq Q$ and $|S| \leqslant k$?*

Problem 5.3.9 (The SSoD enforceability optimization problem) *Given P, R, PA and $Q \subseteq P$, find $S \subseteq R$ such that $\mathsf{Prms}(S) \supseteq Q$ and $|S|$ is minimized.*

Problem 5.3.10 (The RSSoD generation problem) *Given P, R, PA and $Q \subseteq P$, find all $S \subseteq R$ such that $\mathsf{Prms}(S) \supseteq Q$ and for any $S' \subset S$, $\mathsf{Prms}(S') \not\supseteq Q$.*

Note that these questions are only concerned with the existence of covers of Q (and not with any additional permissions that might be authorized for any given cover). Hence, we may simply set $X = Q$ and $\mathcal{C} = \{\mathsf{Prms}(r) \cap Q : r \in \mathcal{S}(Q)\}$. The SSoD enforceability decision problem is, therefore, identical to the set cover decision problem (and hence is **NP**-complete).[6]

The SSoD enforceability optimization problem is of interest for a number of applications. For example, given Q, we may wish to know the smallest number of users that are collectively authorized for Q in order to assess whether this presents some potential violation of enterprise security policies or statutory requirements. It is clear that

[6] Li *et al* showed that the SSoD enforceability decision problem is **NP**-complete by showing that a particular subcase is **NP**-complete [65].

the SSoD enforceability optimization problem is identical to the set cover optimization problem, which is **NP**-hard.

When seeking to enforce an SSoD constraint using SMER constraints, it is necessary to compute the set of RSSoD constraints [65]. More specifically, Li *et al* convert an SSoD constraint into an equivalent set of RSSoD constraints and then find a set of SMER constraints such that the RSSoD constraints (and hence the SSoD constraint) are satisfied if the SMER constraints are satisfied. They proposed a method of generating RSSoD constraints (essentially as described in Problem 5.3.10 above), but provide no analysis of the complexity of computing the set of all such constraints. Note that an RSSoD constraint is a set of roles that cover Q and contains no redundancy. In other words, the RSSoD generation problem is identical to the irreducible cover enumeration problem and is, therefore, **NP**-hard (Theorem 5.2.2). The above results are summarized in the following theorem.

Theorem 5.3.3 *The SSoD enforceability decision problem is **NP**-complete; the SSoD enforceability optimization problem and the RSSoD generation problem are **NP**-hard.*

5.4 Conclusion

In this chapter, we have defined a number of extensions of the standard set cover problem. In particular, we have introduced the notions of container, minimal container, minimal cover, irreducible cover and optimal cover, and established complexity results for a number of problems associated with these notions. Most significantly, we proved that the minimal cover problem – a generalization of the set cover problem – is **NP**-hard.

Our complexity results for the variations on the set cover problem have established the computational complexity of a number of fundamental problems in RBAC: in par-

Problem Name	**Equivalent Set Cover Problem**	**Complexity Class**
Preliminary IDRM	$V \in \mathsf{PCov}$? (that is, $V = \mathsf{ker}(V)$?)	**P**
Exact IDRM decision	Set cover decision	**NP**-complete
Exact IDRM optimization	Set cover optimization	**NP**-hard
IDRM-safety	Compute $\mathsf{ker}(V)$	**P**
IDRM-availability	Minimal cover	**NP**-hard
SSoD enforceability decision	Set cover decision	**NP**-complete
SSoD enforceability optimization	Set cover optimization	**NP**-hard
UAQ	Container optimization	**NP**-hard
RSSoD generation	Irreducible cover enumeration	**NP**-hard

Table 5.1: A summary of problems in RBAC and their computational complexities

ticular, the IDRM-safety and availability problems, the UAQ problem and the RSSoD generation problem. We summarize our results in Table 5.1. Clearly, our immediate priority in future work is to investigate whether our complexity results have other applications in RBAC and other access control models (or indeed other resource allocation problems).

We conclude by noting that the complexity result of the minimal cover problem can be applied to assessing the difficulty of enforcing the principal of least privilege in a general access control model. Let X represent the set of all resource-action pairs and let U represent the set of all users. Then an authorization policy may be modeled as a function $p : U \to 2^X$, where $p(u) \subseteq X$ denotes the interactions for which u is authorized. Given $V \subseteq U$, we will abuse our notation and write $p(V)$ to denote the interactions for which the users in V are collectively authorized.

The principles of *least privilege*, defined by Saltzer and Schroeder [77], is of particular importance in access control. Informally, least privilege means that users should be given no more access to resources than is required for those users to perform their duties. Given a set of resource-action pairs $Y \subseteq X$, what is the best choice of users to cover Y with respect to least privilege. We formally consider the question in the following form.

Problem 5.4.1 (The least privilege problem) *Given a policy p and $Y \subseteq X$, find a subset of users V such that $p(V) \supseteq Y$ and $|p(V)|$ is minimized.*

Given any instance $(X, \mathcal{C}, V)$ of the minimal cover problem, we can simply set $Y = V$, and for each $C \in \mathcal{C}$, define $u_C \in U$ and $p(u_C) = C \subseteq X$. The least privilege problem is, therefore, identical to the minimal cover problem (and hence is **NP**-hard). In the next chapter, we will study some heuristic algorithms for solving the minimal cover problem.

Chapter 6

Heuristic Algorithms for the Minimal Cover Problem

In the preceding chapter we observed that many relevant optimization problems are **NP**-hard, so it is unlikely that we will ever be able to find efficient algorithms for their exact solution. However, we can focus on finding an algorithm that runs in polynomial time and that has the property that, for every problem instance, the solution returned by the algorithm for that instance is "close" to the exact solution. We call such algorithms *heuristic algorithms*, and the solutions returned by those algorithms as *approximate solutions*.

The *quality* of a heuristic algorithm is measured by analyzing the "distance" of its solution to the exact solution for each instance, which we would like to be as small as possible. The most common approach of measuring the quality of a heuristic algorithm is to conduct empirical experiments on a large set of instances that can be either randomly generated or based on instances arising in real problems, and then establish the average-case performance of the algorithm on this set [5]. From a theoretical point of view, we might be interested in finding a heuristic algorithm that returns solutions

with guaranteed performance: that is, the solutions returned by the algorithm for all instances will never differ from the exact solution by more than some specified percentage. These heuristic algorithms are usually called *approximation algorithms* in the literature.

In Chapter 5, we showed that the minimal cover problem is **NP**-hard. In other words, there does not exist a polynomial time (efficient) algorithm that can be used to solve all instances of the problem. Therefore, we aim to seek a good heuristic algorithm to solve the minimal cover problem. In this chapter, we design a number of polynomial time algorithms that compute approximate solutions to the minimal cover problem. We also conduct a number of experiments to evaluate the quality of those algorithms in terms of the average case performance.

This chapter is organized as follows. In the next section, we introduce a "greedy" algorithm for the weighted set cover optimization problem, which provides the basis for the design of our heuristic algorithms for the minimal cover problem. Section 6.2 informally describes a number of ways of designing and evaluating heuristic algorithms for the minimal cover problem. In Section 6.3 we conduct some experiments to evaluate the average case performance of the heuristic algorithms, and analyze the results of our experiments.

6.1 The weighted set cover optimization problem

In this section, we introduce the weighted set cover optimization problem, and explain the basic structure of the greedy algorithm that is a good heuristic algorithm for the weighted set cover problem. This, in turn, provides a basic starting point in the design of heuristic algorithms for the minimal cover problem.

We formally define the weighted set cover optimization problem in the following

form.

Problem 6.1.1 (The weighted set cover optimization problem) *Given a universe X, a collection $\mathcal{C}$ of subsets of X whose union is X, and a weight function $w : \mathcal{C} \to \mathbb{R}^+$, find a subset $\mathcal{D} \subseteq \mathcal{C}$ such that*

$$X = \bigcup_{D \in \mathcal{D}} D \quad \textit{and} \quad \sum_{D \in \mathcal{D}} w(D) \quad \textit{is minimized.}$$

Although the weighted set cover optimization problem is **NP**-hard[1], there exists a "greedy" algorithm that provides good approximate solutions [24, 53]. This iterative algorithm sequentially selects elements from $\mathcal{C}$ until all the elements of X are covered. Suppose we have chosen $i-1$ sets from $\mathcal{C}$, and let $X_{i-1} \subseteq X$ denote set of elements of X that remain uncovered after the $(i-1)$th iteration. At the ith iteration, the greedy algorithm selects a subset C_i from those remaining in $\mathcal{C}$ such that

$$score(C_i) = \frac{w(C_i)}{|C_i \cap X_{i-1}|}$$

is minimized. In other words, at each iteration, the greedy algorithm computes a score for each $C \in \mathcal{C}$ and selects the element of $\mathcal{C}$ with the smallest score. The score is obtained by dividing the "cost" of selecting C by the "benefit", where the cost is defined to be the weight of C and the benefit is the number of uncovered elements that C covers.

The special case where the weight of C is constant for all $C \in \mathcal{C}$ corresponds to the standard set cover optimization problem. We can still apply the greedy algorithm to solve the set cover optimization problem. At each iteration, the algorithm selects an

[1]The set cover optimization problem is a subcase of the weighted set cover optimization problem, where the weight of $C \in \mathcal{C}$ is constant. Therefore, there exists a Turing reduction from the set cover optimization problem to the weighted set cover optimization problem, and, since the set cover optimization problem is **NP**-hard, the weighted set cover optimization problem is **NP**-hard.

element C of $\mathcal{C}$ with smallest score, where the cost of selecting C is constant.

It has been shown that the greedy algorithm is the best-possible polynomial time approximation algorithm for the weighted set cover optimization problem [91]. The weight of the solution produced by the greedy algorithm is guaranteed to be no more than $h(n)$ times the weight of the optimal solution, where $h(n)$ is the nth harmonic number and n is the cardinality of X[2].

6.2 Designing heuristic algorithms

In this section, we consider a number of possible ways to design heuristic algorithms for the minimal cover problem. We also discuss two approaches for evaluating average-case performance of a heuristic algorithm.

6.2.1 Designing an algorithm

The greedy algorithm for the weighted set cover optimization problem is designed to compute a cover of X and to minimize the weights of sets used in the cover. When computing a minimal cover, we are concerned with computing a cover of V *and* minimizing the number of elements outside V that are covered. Using an approach similar to the greedy algorithm for the weighted set cover optimization problem, we can devise an algorithm that sequentially selects elements of $\mathcal{C}$ until all the elements of V are covered. We define the benefit of C to be $|C \cap V_{i-1}|$, where V_{i-1} comprises the members of V that remain uncovered after the $(i-1)$th iteration (as in the algorithm for the weighted set cover optimization problem). However, we need to consider which C is most appropriate to be selected at each iteration. This is determined by three factors: how to appropriately define the cost of C, how to compute the cost of C at

[2]The nth harmonic number is the sum of the reciprocals of the first n natural numbers, that is, $h(n) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k}$.

each iteration, and how to balance the trade-off between cost and benefit of C at each iteration. We now consider three factors in more detail.

Defining the cost

One obvious possibility is to define the cost of C to be the number of elements in C multiplied by the number of elements in C but not in V. In this case, the cost of C become very large if C includes a large number of elements that are both in V and outside V. Clearly, the benefit of such C is also large, which in turn, means the score of such C is correspondingly large. In other words, this approach of defining the cost of C might provide a more direct reflection on the number of elements outside V that C contains.

Alternatively, we could define the cost of C to be the number of elements that are in C but not in V. In other words, the cost of C is determined by the extent to which C is a "bad fit" for the elements that need to be covered.

Finally, we could define the cost of C to be the sum of the weights of the elements in C that are not in V, where the "weight" of $x \in C \setminus V$ is the reciprocal of the number of elements of $\mathcal{C}$ in which x appears. The intuition here is that if x has a low weight, it belongs to many elements of $\mathcal{C}$, which makes it likely that it will have to be included in any cover of V. In this case, C will have a low cost if most of its elements are in V and the reminder belong to many other elements of $\mathcal{C}$.

Example 6.2.1 *Let* $X = \{1, 2, 3, 4\}$ *and let* $\mathcal{C} = \{C_1, C_2, C_3, C_4\}$*, where* $C_1 = \{1\}$*,* $C_2 = \{2, 4\}$*,* $C_3 = \{3, 4\}$ *and* $C_4 = \{1, 2, 4\}$*. Let* $V = \{1, 2, 3\}$*. Consider* $C_4 = \{1, 2, 4\}$ *that has three elements and one element* 4 *outside* V *which also appears in* C_2 *and* C_3*. Then, the cost of* C_4 *is* $3 \times 1 = 3$ *if using the first way of defining the cost. In turn, the cost of* C_4 *is* 1 *if using the second way, and the cost of* C_4 *is* $\frac{1}{3}$ *if the third way is used.*

Modifying the target

At each iteration of an algorithm we will add a new $C \in \mathcal{C}$, and C may contain an element $x \notin V$. At the next iteration, therefore, it does not matter if we select $C' \in \mathcal{C}$ such that $x \in C'$; we are not adding any additional elements that do not belong to V. Hence there are two different ways in which we may compute the cost of C: based on V or based on the union of V and the sets already selected for the cover. For brevity, we refer to these alternatives as using a static or dynamic "target", respectively. Clearly, the third way of defining the cost of C (described above) is only likely to be effective if we use dynamic targets.

Computing a score

At each iteration, if we attempt to select $C \in \mathcal{C}$ such that the score of C is minimized, then there are two different ways in which we can combine the cost and benefit to compute a score for C: we could divide cost by benefit or we could subtract the benefit from the cost. Another alternative is to ignore the benefit and focus on minimizing the cost. If two distinct elements of $\mathcal{C}$ have the same cost, then we select the one with the greater benefit.

6.2.2 Evaluation metrics

We have seen that there are a number of possible scoring functions that we could use for designing heuristic algorithms for the minimal cover problem. We now consider how we might evaluate these candidate algorithms.

Given a heuristic algorithm for the minimal cover problem, there are (at least) two ways we could measure the average case performance of the heuristic algorithm. We define the following metrics: success rate and mean deviation.

Definition 6.2.1 *Suppose we have m instances of the minimal cover problem $\{(X, \mathcal{C}_i, V) : 1 \leqslant i \leqslant m\}$, where $|X|$ and $|V|$ are fixed. Let $\mathcal{S}_i$ denote the solution computed by algorithm A for problem instance i and let $\mathcal{D}_i$ denote a minimal cover for problem instance i. Then we define:*

- *the* success rate *of A to be*

$$\frac{1}{m} |\{i \in \{1, \ldots, m\} : |U_{\mathcal{S}_i}| = |U_{\mathcal{D}_i}|\}|$$

- *the* mean deviation *of A to be*

$$\frac{1}{m} \sum_{i=1}^{m} (|U_{\mathcal{S}_i}| - |U_{\mathcal{D}_i}|)$$

In other words, given a finite number m of instances of the minimal cover problem, the success rate of a heuristic algorithms indicates the proportion of exact solutions computed by the algorithm, while the mean deviation of a heuristic algorithm indicates the average of the difference of the solutions returned by the algorithm and the exact solution in terms of the number of elements outside V.

6.3 Evaluating heuristic algorithms

In order to find a good heuristic algorithm for the minimal cover problem, we conducted a number of experiments to analyze the average case performance of a number of heuristic algorithms derived from the design options described in the previous section. Firstly, we generate a set of appropriate test data: that is, given a universe $X = \{1, 2, \ldots, 10\}$, we generate 10000 different instances of $\mathcal{C}$. We also select a number of different choices of $V \subseteq X$ where $3 \leqslant |V| \leqslant 7$. Finally, we implement 18 different

heuristic algorithms. For each $\mathcal{C}$ and each V, we apply each of these heuristic algorithms and compute the resulting cover. We then analyze the results of our experiments.

6.3.1 Data generation

We model a collection $\mathcal{C}$ of subsets of X as a $k \times n$ matrix, where $|\mathcal{C}| = k$ and $|X| = n$. That is, $\mathcal{C}[i, j] = 1$ if and only if the jth element of C_i belongs to X.

Populating the matrix

We generate the elements of a $k \times n$ array by deciding at random which elements to set to 1. We select a value q, $0 < q < 1$, as the probability that an array element is set to 0. Then, for each element of the array $\mathcal{C}[i, j]$, we select r between 0 and 1 uniformly at random and set $\mathcal{C}[i, j] = 0$ if $r \leqslant q$ and $\mathcal{C}[i, j] = 1$ otherwise.

We choose q so that the probability of obtaining a row or column of zeroes is smaller than some threshold t. In particular, if $m = \min\{k, n\}$, then we require that $q^m \leqslant t$: that is, $m \log_e q \leqslant \log_e t$. Hence, we choose a value of q such that

$$q \leqslant e^{\frac{\log_e t}{m}}.$$

If, for example, $k = n = 10$ and $t = 0.001$, then we choose $q = 0.50$ (correct to two decimal places). If, in contrast, $k = n = 10$ and $t = 0.01$, we choose $q = 0.63$. The proportion of 1s in the matrix increases with $p = 1 - q$ and hence the number of 1s, on average, in row j (which corresponds to the cardinality of C_j) increases as q decreases.

In the experiment, we set X to be $\{1, 2, \ldots, 10\}$, and randomly choose an element from $\{5, 6, \ldots, 15\}$ for the cardinality k of $\mathcal{C}$. Hence $\min\{|\mathcal{C}|, |X|\} = 10$. We set $q = 0.63$ so that the generated $k \times n$ array has probability no greater than 0.01 of obtaining a row or column of zeroes.

Discarding matrices

Having generated a matrix $\mathcal{C}$, we discard it if one of the following conditions is satisfied:

- It contains a row of zeroes. In this case, one of the elements of the collection of subsets represented by $\mathcal{C}$ contains the empty set.
- It contains a column of zeroes. In this case, an element of X is not covered by any element of $\mathcal{C}$.
- We have already generated a matrix $\mathcal{C}' = \mathcal{C}$. That is, we remove duplicates.

Each time we generate a matrix that is not discarded for one of the above reasons, we increment a counter. Once we had generated 10000 matrices we stopped.

6.3.2 Scoring functions

Like the greedy algorithm for the weighted set cover optimization problem, each of our heuristic algorithms sequentially selects elements from $\mathcal{C}$ until V is covered. At each iteration, we select $C \in \mathcal{C}$ such that the *score* of C is minimized, where the score of C is computed from the cost and benefit associated with C. We define the benefit function to be $\mathit{benefit} : \mathcal{C} \to \mathbb{R}^+$, where $\mathit{benefit}(C)$ is defined to be $|U_i \cap C|$. Here U_i denotes the set of elements in V that remain uncovered before iteration i. That is, $U_1 = V$ and if the algorithm selects $C \in \mathcal{C}$ at iteration i, then $U_{i+1} = U_i \setminus C$.

There are three different ways in which the cost function $\mathit{cost} : \mathcal{C} \times 2^X \to \mathbb{R}^+$ can be defined (as discussed informally in Section 6.2).

1. $\mathit{cost}_1(C, V) = |C| \cdot |C \setminus V|$
2. $\mathit{cost}_2(C, V) = |C \setminus V|$
3. Let $f(x)$, $x \in X$, be the number of elements in $\mathcal{C}$ that contain x; that is $f(x) =$

$|\{C \in \mathcal{C} : x \in C\}|$. Then define

$$cost_3(C, V) = \begin{cases} 0 & \text{if } C \subseteq V \\ \sum_{x \in C \setminus V} \frac{1}{f(x)} & \text{otherwise} \end{cases}$$

Now for each cost function, we may assume that the cost of each element in $\mathcal{C}$ is static during the execution of the algorithm. A natural alternative is to re-compute the cost of each remaining member of $\mathcal{C}$ at each iteration. For example, we define $cost_2(C, V_i) = |C \setminus V_i|$, where V_i denotes the set of elements that are permitted to belong to the cover at iteration i. That is, $V_1 = V$ and if the algorithm selects $C \in \mathcal{C}$ at iteration i, then $V_{i+1} = V_i \cup C$. The advantage of recomputing the cost of each remaining C is that in choosing C, we expand V to $V \cup C$, and it may be that we can choose C' to cover other elements of V without including any elements outside $V \cup C$. Therefore, we may compute each cost function based on a static $V = V_i = V_{i+1}$ for all i or dynamic V_i (where $V_1 = V$ and $V_{i+1} = V_i \cup C_i$).

At each iteration, if we want to simultaneously minimize cost and maximize benefit, there are two different ways in which we combine $cost(C, V)$ and $benefit(C)$, that is $cost(C, V)/benefit(C)$ and $cost(C, V) - benefit(C)$. In addition, we could consider cost on its own. If there is more than one $C \in \mathcal{C}$ having minimal cost, we then select the one with maximum benefit. In other words, the benefit function is used to assist the algorithm to select a more appropriate $C \in \mathcal{C}$ when it is necessary. We refer to this combining method as *benefit-assisted-cost*.

In summary, there are 18 different possible scoring functions. We write $score_{ijk}$, where $1 \leqslant i \leqslant 3$, $1 \leqslant j \leqslant 3$ and $1 \leqslant k \leqslant 2$, to denote the different scoring functions for each $C \in \mathcal{C}$. Each scoring function $score_{ijk}$ is distinguished by assigning corresponding

values to i, j and k, which indicates the use of different cost functions, different combining methods, and different ways of computing cost respectively. More specifically, we explain the meanings for each possible values of i, j and k as follows.

$i = 1$ denotes $cost_1$ function
$i = 2$ denotes $cost_2$ function
$i = 3$ denotes $cost_3$ function
$j = 1$ denotes division combining method
$j = 2$ denotes subtraction combining method
$j = 3$ denotes benefit-assisted-cost combining method
$k = 1$ denotes dynamic V
$k = 2$ denotes static V

For example, those scoring functions that use $cost_1$ have the form $score_{1jk}$ and are listed below.

$$score_{111}(C) = \frac{cost_1(C, V_i)}{benefit(C)}$$

$$score_{112}(C) = \frac{cost_1(C, V)}{benefit(C)}$$

$$score_{121}(C) = cost_1(C, V_i) - benefit(C)$$

$$score_{122}(C) = cost_1(C, V) - benefit(C)$$

$$score_{131}(C) = cost_1(C, V_i)$$

$$score_{132}(C) = cost_1(C, V)$$

The scoring functions $score_{ijk}$, $i \in \{2, 3\}$, $j \in \{1, 2, 3\}$ and $k \in \{1, 2\}$ are defined in an analogous way. We will write alg_{ijk} to denote the heuristic algorithm that employs scoring function $score_{ijk}$.

6.3.3 Results

Now we have $X = \{1, 2, \ldots, 10\}$ and 10000 distinct test cases $\mathcal{C}$. We choose a number of different $V \subset X$ such that $3 \leqslant |V| \leqslant 7$. For each $\mathcal{C}$ and each V, we apply each of

18 different heuristic algorithms to compute a resulting cover, and apply a brute-force algorithm that considers every possible subset of $\mathcal{C}$ to compute a minimal cover.

Table 6.1 summarizes the results for the best three heuristic algorithms for different choices of $|V|$. Each row in the table is associated with a choice of V, and indicates the best three heuristic algorithms for that choice of V. The table is divided into three sections for the first, second and third best algorithms. Each section has three columns: the identity of the heuristic algorithm (id), the success rate of the heuristic algorithm s, and the mean deviation of the heuristic algorithm d.

$\|V\|$	First			Second			Third		
	id	s	d	id	s	d	id	s	d
3	alg_{211}	88.33%	0.1244	alg_{221}	87.39%	0.1373	alg_{311}	87.12%	0.1402
4	alg_{211}	88.41%	0.1235	alg_{311}	88.17%	0.1250	alg_{221}	87.31%	0.1360
5	alg_{311}	89.91%	0.1054	alg_{211}	89.48%	0.1109	alg_{221}	87.78%	0.1292
6	alg_{311}	92.82%	0.0733	alg_{211}	91.86%	0.0830	alg_{221}	87.48%	0.1307
7	alg_{311}	95.74%	0.0428	alg_{211}	94.42%	0.0563	alg_{111}	91.32%	0.0884

Table 6.1: The results of the best heuristic algorithms for different $|V|$

We make a number of observations about the results reported in Table 6.1.

- Any heuristic algorithm with static V (that is, $k = 2$ in alg_{ijk}) is always worse than any heuristic algorithm with dynamic V ($k = 1$). It can be seen from Table 6.1 that no heuristic algorithm with static V appears in the table.

- As $|V|$ increases, the probability of computing the minimal cover for each heuristic algorithm increases, and the sum of the distances (which yields the mean deviation) for each heuristic algorithm decreases. Intuitively, as the number of elements outside V decreases, each heuristic algorithm is more likely to compute the minimal cover, and the mean deviation of each heuristic algorithm decreases.

- For $|V| = 3$ or $|V| = 4$, the heuristic algorithm alg_{211} is best; as $|V|$ increases,

the best heuristic algorithm becomes alg_{311}.

Table 6.2 summarizes the results for dynamic heuristic algorithms when $|V| = 5$. Each row is labelled with a cost function and each column is labelled with a method for combining cost and benefit (division, subtraction or assistance). An entry in row $cost_i$ and column $combine_j$, indicates the success rate and the mean deviation for the heuristic algorithm constructed by $cost_i$, $combine_j$ (and V is re-computed at each iteration).

Cost function	Division		Subtraction		Assistance	
1	84.19%	0.1684	72.47%	0.3132	64.72%	0.4203
2	89.48%	0.1109	87.78%	0.1292	80.96%	0.2102
3	89.91%	0.1054	76.47%	0.3044	78.38%	0.2397

Table 6.2: The results for dynamic heuristic algorithms when $|V| = 5$

Table 6.2 has some interesting features, which we summarize below.

- Division is always better than subtraction and assistance when using the same cost function.
- Subtraction is better than assistance for $cost_1$ and $cost_2$, but not for $cost_3$.
- A dynamic heuristic algorithm with assistance is always worse than a dynamic heuristic algorithm with division (irrespective of the cost function).

Therefore, these results suggest that we should use division as a method of combining cost and benefit and use dynamic way of computing cost in the design of a good heuristic algorithm.

6.3.4 A hybrid algorithm

In the previous section, we identified two heuristic algorithms that performed better than all others: alg_{211} and alg_{311}. In particular, alg_{211} is the best heuristic algorithm

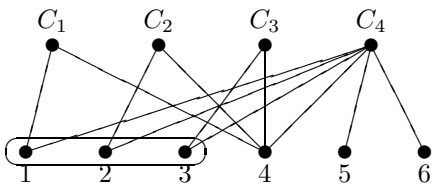
when $|V|$ is small; as $|V|$ increases, alg_{311} becomes the best. Let us now try to identify situations when these two heuristic algorithms do not perform well. This enables us to examine the functioning of each of these two algorithms, and leads to ideas for obtaining algorithms with improved performance no matter which V is used.

Algorithm alg_{211}

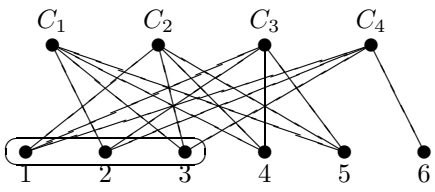
Consider the following example: let $X = \{1, 2, \ldots, 2m\}$, $\mathcal{C} = \{C_1, \ldots, C_{m+1}\}$ and $V = \{1, \ldots, m\}$, where $C_i = \{i, m+1\}$, $1 \leqslant i \leqslant m$, and $C_{m+1} = X$. Figure 6.1(a) illustrates a graphical representation of this example for $m = 3$, where the closed curve indicates $V = \{1, 2, 3\}$. Now we can see that

$$score_{211}(C_1) = \cdots = score_{211}(C_m) = \frac{1}{1} = score_{211}(C_{m+1}) = \frac{m}{m} = 1.$$

In the first iteration, alg_{211} might therefore choose C_{m+1}, and then terminate. In this case, the cover contains m elements that do not belong to V, whereas a minimal cover $\{C_1, \ldots, C_m\}$ contains only one element that is not in V. However, the heuristic algorithm alg_{311} is able to compute the minimal cover $\{C_1, \ldots, C_m\}$, because $cost_3$ is designed to select those C whose elements outside V also belong to other members of $\mathcal{C}$.



(a) An example for alg_{211}



(b) An example for alg_{311}

Figure 6.1: A graphical representation of the minimal cover problem instances

Algorithm alg_{311}

Now consider this example: let $X = \{1, \ldots, 2m\}$, $V = \{1, \ldots, m\}$ and $\mathcal{C} = \{C_1, \ldots, C_{m+1}\}$, where $C_i = \{1, \ldots, i-1, i+1, \ldots, m, m+1, m+2, \ldots, 2m-1\}$, $1 \leqslant i \leqslant m$, and $C_{m+1} = \{1, \ldots, m, 2m\}$. Figure 6.1(b) illustrates a graphical representation of this example for $m = 3$, where the closed curve indicates $V = \{1, 2, 3\}$. Then

$$score_{311}(C_1) = \cdots = score_{311}(C_m) = \frac{1}{m-1}\frac{m-1}{m} = \frac{1}{m} = score_{311}(C_{m+1}) = \frac{1}{m}.$$

At the first iteration, alg_{311} might choose C_1, then select C_2 at the next iteration, then C_3, ..., finally selecting C_m before terminating. The resulting cover contains $m-1$ elements that are not in V. However, the minimal cover is $\{C_{m+1}\}$, which contains only one element that is not in V. In this example, alg_{311} is not able to choose those C which have fewest elements outside V, while alg_{211} can compute the optimal solution $\{C_{m+1}\}$.

Combining alg_{211} **and** alg_{311}

We have illustrated an infinite family of instances in which the solution by each of alg_{211} and alg_{311} might be far away from the exact one. However, there is an interesting characteristic of these two algorithms: when one of them, for example alg_{211} does not perform very well for some instances, the other one, alg_{311}, is always able to obtain a minimal cover for those instances. Therefore, we consider a new scoring function that can provide an appropriate combination of $score_{211}$ and $score_{311}$, and always stand out with the best performance no matter what size of V is used. We propose a scoring

function that takes the average of $score_{211}$ and $score_{311}$, that is

$$score_{411}(C) = \frac{1}{2}\left(score_{211}(C) + score_{311}(C)\right)$$

We implement the heuristic algorithm alg_{411}, and evaluate the quality of alg_{411} by comparing with alg_{211} and alg_{311}. Table 6.3 summarizes the results for alg_{211}, alg_{221}, alg_{311} and alg_{411}. It can be seen that alg_{411} is always best no matter which V is chosen.

$\|V\|$	First			Second			Third		
	id	s	d	id	s	d	id	s	d
3	alg_{411}	90.21%	0.1026	alg_{211}	88.33%	0.1244	alg_{221}	87.39%	0.1373
4	alg_{411}	90.45%	0.0994	alg_{211}	88.41%	0.1235	alg_{311}	88.17%	0.1250
5	alg_{411}	91.58%	0.0874	alg_{311}	89.91%	0.1054	alg_{211}	89.48%	0.1109
6	alg_{411}	94.09%	0.0597	alg_{311}	92.82%	0.0733	alg_{211}	91.86%	0.0830
7	alg_{411}	96.24%	0.0377	alg_{311}	95.74%	0.0428	alg_{211}	94.42%	0.0563

Table 6.3: The results for alg_{211}, alg_{311} and alg_{411}

Algorithm alg_{411}

We also believe that the performance of algorithm alg_{411} compares favorably with other algorithms that compute approximate solutions for other **NP**-hard problems. As we discussed in Section 6.2, there is a greedy algorithm for the set cover optimization problem that is known to compute a good approximate solution. Indeed, the cardinality of the solution it computes is never more than $h(n)$ times the cardinality of the optimal solution, where $h(n)$ is the nth harmonic number and n is the cardinality of X [24].

Recall that we generated 10000 different instances of $\mathcal{C}$ for $X = \{1, 2, \ldots, 10\}$, which can be also used as the test data for the set cover optimization problem. We ran the greedy algorithm to compute an approximate set cover and an exhaustive search to compute the optimal set cover. We found that the greedy algorithm computed the optimal set cover 87.36% of the time. In contrast, for the minimal cover problem,

alg_{411} had a success rate of 90.21% for $|V| = 3$, rising to 96.24% for $|V| = 7$. In other words, we have grounds for believing that alg_{411} is a good heuristic algorithm for the minimal cover problem.

6.4 Concluding remarks

As the minimal cover problem is **NP**-hard, we developed 18 different heuristic algorithms using three different cost functions, two different ways of computing the cost function at each iteration, and three different ways of combining cost and benefit at each iteration. We conducted some experiments to evaluate the average case performance of each of these algorithms. More specifically, we generated 10000 different instances of C for $X = \{1, 2, \ldots, 10\}$, and varied the cardinality of V to compute the success rate and the mean deviation of each of algorithm. Our results show that the heuristic algorithms alg_{211} and alg_{311} have a better performance than the other algorithms, but alg_{211} and alg_{311} have different performance when the cardinality of V is chosen from small to large.

Then we defined a new heuristic algorithm alg_{411} with a new scoring function $score_{411}$ that combines the scores ($score_{211}$ and $score_{311}$) of the two best algorithms. Our results show that the heuristic algorithm alg_{411} has the best performance, irrespective of the choice of V.

An interesting possibility in future work is to formally examine why alg_{211} and alg_{311} have different performances when $|V|$ is chosen differently. We believe that the answer to this question is useful to establish an *approximation ratio* for the best algorithm alg_{411} we obtained in the experimental work in this chapter. In particular, we hope to establish a bound for the ratio

$$\frac{|U_{\mathcal{S}_i}| - |V|}{|U_{\mathcal{D}_i}| - |V|}$$

where $\mathcal{S}_i$ is the cover computed by alg_{411} for instance i and $\mathcal{D}_i$ is the minimal cover of instance i (see the work of Johnson [54], Chvatal [24] and Feige [37] on the set cover problem, for example).

Chapter 7

Conclusions and Future Work

Broadly speaking, the overall contribution of this thesis is to address several issues related to the use of role hierarchies in role-based access control. We have explored applications of the OP-RBAC model in order to address the perceived deficiencies of inheritance within a role hierarchy. We have also developed new advanced RBAC models that accurately capture the relationship between usage and activation hierarchies, and the interactions between spatio-temporal constraints and a role hierarchy. Furthermore, we have defined and studied a number of important computational problems that could arise in various RBAC models, including those models we developed for multiple role hierarchies and spatio-temporal constraints.

In the next section, we present the contributions of this thesis in more detail, and outline some directions for future work in Section 7.2.

7.1 Summary of contributions

In Chapter 3 we considered three applications of the OP-RBAC model developed by Crampton [29]. The most important innovation in the OP-RBAC model is that permissions can be inherited in one of three ways within the role hierarchy: by more

senior roles, by less senior roles and by no other roles. These permissions are called up permissions, down permissions and neutral permissions respectively. This approach to permission inheritance has some correspondence with the Bell-LaPadula model, where up permissions correspond to read-type access rights in the Bell-LaPadula model, down permissions correspond to append-type access rights and neutral permissions correspond to write-type access rights. We demonstrated OP-RBAC provides a natural way of simulating a number of Bell-LaPadula models with a single role hierarchy, hence making it simpler than existing approaches to this topic [69, 72, 73, 81]. In addition, we show that OP-RBAC can be constrained to support the assignment of compound permissions (write permissions) and incorporate limited support for discretionary access control policies of the Bell-LaPadula model. We believe no existing work attempts to address these two features in role-based models. Furthermore, we introduced a new variation of the Bell-LaPadula model in which every object is associated with two security labels, one governing read access and one governing append access. This model is useful to provide suitable access control for an audit file, for example. We also illustrated that OP-RBAC can simply implement this new version of the Bell-LaPadula model with the addition of few constraints.

The second application of the OP-RBAC model was shown to implement dynamic separation of duty constraints on two roles that have a common senior role and for a user to activate the senior role. We may require that all sensitive permissions assigned to the two junior roles are neutral permissions, and a user who activated the senior role is not able to acquire the sensitive permissions. This is not possible in standard RBAC models.

Finally, we constructed an OP-RBAC configuration that is equivalent to an ERBAC96 configuration with respect to the same access requests are authorized. The similarity between ERBAC96 and OP-RBAC arises because both models use the same

role hierarchy for role activation, and OP-RBAC can make use of up and neutral permissions to implement permission usage in ERBAC96. Of course, the motivation of this application is to illustrate the power and flexibility of OP-RBAC.

In Chapter 4 we developed a novel extended RBAC model (ERBAC07) based on RBAC_1. Compared with existing work, such as ERBAC96 and GTRBAC, ERBAC07 proposes a new constraint on the relationship between role activation and permission usage hierarchies, which enables ERBAC07 to have the most appropriate inheritance semantics. We also introduced a graph-based formulation of RBAC_1, which is useful in determining which access requests are authorized. We then extended this graph-based formulation to explain the authorization semantics of ERBAC07. These graph-based formalisms provide the basis for the semantics of our spatio-temporal RBAC models.

We defined three spatio-temporal RBAC models: $\text{RBAC}_{ST}^{=}$, RBAC_{ST}^{+}, and RBAC_{ST}^{-}, motivating the development of these models using simple scenarios. These models extend the basic RBAC_1 model with very little additional syntax, have clear, well-defined authorization semantics, and are designed to be compatible with RBAC_1 and the ANSI-RBAC standard. We also introduced the concept of trusted entities on each of three spatio-temporal RBAC models: for such entities spatio-temporal constraints may be ignored, in order to deal with certain scenarios. In addition, we developed three spaio-temporal ERBAC models to support spatio-temporal requirements in ERBAC07. We believe that our spatio-temporal RBAC and ERBAC models are simpler, more expressive and more flexible than existing spatio-temporal models.

Equally important, we considered the implementation of our spatio-temporal RBAC models in practical applications, which have not previously been investigated for any existing spatio-temporal models. The existence of spatio-temporal constraints and a role hierarchy may result in complex computations when checking access requests. We proposed a way of improving the efficiency of access request checking by pre-computing

spatio-temporal enabling conditions over the transitive closure of (part of) the RBAC graph. In addition, we proved that flat RBAC^{+}_{ST} is able to encode most spatio-temporal requirements that defined on $\text{RBAC}^{=}_{ST}$, RBAC^{+}_{ST} and RBAC^{-}_{ST}, although this comes at the cost of larger *UA* and *PA* relations. Nevertheless, request evaluation time, which is more important, is likely to be significantly reduced in a flat RBAC^{+}_{ST} system. On the other hand, if a role hierarchy is required, we showed how to eliminate spatio-temporal constraints on roles by limiting constraints to outer nodes and edges, such as users and user-role assignments. In short, we believe that our models can be efficiently and easily implemented, in contrast to existing models.

In Chapter 5 we developed a mathematical framework that provides a context for identifying a number of computational problems that are variations on the standard set cover problem. The motivation for studying these variations is to formally establish the complexity results for a number of important problems in RBAC. In particular, we defined the minimal cover problem that is a generalization of the set cover problem, and proved that this problem is **NP**-hard. This complexity result is then used to determine the complexity of the IDRM-availability problem and the user authorization query problem. In addition, we introduced the irreducible cover problem that is a restricted form of the set cover problem. We proved that the irreducible cover problem is **NP**-hard, which in turn establishes the complexity of the RSSoD generation problem.

In Chapter 6 we designed a number of heuristic algorithms to answer the minimal cover problem based on the ideas of the greedy algorithm for the weighted set cover optimization problem. We conducted experiments to evaluate the average performance of these algorithms. From our experiments, we found that one of the heuristic algorithms, alg_{411}, has the best performance with at least 90% rate of computing an exact solution to the minimal cover problem. In summary, we believe that Chapter 5 and 6 make significant contributions to the understanding and practical solution of the minimal

cover problem.

7.2 Future work

There are many opportunities for extending the work presented in this thesis. In the following section we will outline these extensions.

In Chapter 3, OP-RBAC is illustrated to provide a direct implementation of the Bell-LaPadula models with the addition and modification of a few constraints. The natural extension of the ideas in Chapter 3 is to develop a general multi-level secure model based on OP-RBAC with the consideration of trusted subjects and complex permissions. The basic idea is to enable the general model to combine the strong security properties of the Bell-LaPadula model with the flexibility of the RBAC model. In other words, we would like to develop formal statement of the security properties of the general model when constrained in the ways described in Chapter 3, and attempt to develop a result that is analogous to the "Basic Security Theorem" for the Bell-LaPadula model [12]. The next phase of the work will be to implement the general model in a prototype system that provides an opportunity to evaluate the suitability of the model.

In Chapter 4 we developed a novel ERBAC07 model on which there are two interesting directions for future work. In Chapter 5 we examined the complexity of generation of SMER constraints on RBAC96 to enforce a SSoD policy. The complexity arises when converting the SSoD policy into a equivalent set of RSSoD constraints. We would like to investigate a simpler approach to this problem by directly generating appropriate constraints on the usage hierarchy of ERBAC07 to enforce the SSoD policy. In addition, it has been suggested that a desirable feature in GTRBAC is to define "upward delegation" in multiple role hierarchies, which allows a user to delegate a per-

mission to roles more senior than the role to which the permission is assigned [55]. We would like to extend existing delegation models [7, 33, 95] or develop a new delegation model for ERBAC07 to deal with complex delegation operations through multiple role hierarchies.

In Chapter 4 we also constructed a number of spatio-temporal role-based models and discussed the use of these models in practice. A first priority of future work on those spatio-temporal models is to investigate spatio-temporal separation of duty. We would like to formally classify various spatio-temporal separation of duty constraints, and propose efficient mechanisms for enforcing those constraints. For example, consider a spatio-temporal separation of duty constraint that specifies no user is allowed to activate roles r_1 and r_2 at any spatio-temporal domain $\mathcal{D}$. We can simply enforce this constraint in $\text{RBAC}^{=}_{ST}$ by defining that $\lambda(r_1) \cap \lambda(r_2) = \emptyset$. In other words, we might be able to use spatio-temporal constraints as a mechanism for enforcing spatio-temporal separation of duty constraints. These questions will occupy our research in the short term.

Another interesting research direction of the work in Chapter 4 is to extend the model to any partially ordered set of entity attributes, not just space and time. For example, imagine that there are several security domains within an organization and that each domain is associated with a security clearance. Then some entities or entity assignments are only enabled when the user belongs to an appropriate domain. Additionally, we would like to study spatio-temporal requirements in a workflow system, and plan to extend our spatio-temporal models to support such systems.

In Chapter 5 we established complexity results for the minimal cover problem and some auxiliary problems. An immediate priority in future work is to investigate whether these complexity results can be applied to some other problems in the context of RBAC, for example, the role mining problem [61, 90].

A further potential area of research is to identify more interesting problems and establish their complexity results within the framework we built in Chapter 5. Recall that the concept of minimal container is concerned with minimizing the number of elements outside V that are included in any container of V. Similarly, we might be interesting to find a least upper bound of V in PCov with respect to subset inclusion. More formally, we have the following definition.

Definition 7.2.1 *Given X, $\mathcal{C}$ and $V \subseteq X$ such that $V \notin \mathsf{PCov}$, we say $T \in \mathsf{PCov}$ is an* irreducible container *of V if T is a container of V and for all $T' \in \mathsf{PCov}$ such that $T' \subset T$, $T' \not\supseteq V$.*[1]

Proposition 7.2.1 *Any minimal container is an irreducible container.*

Proof Let T be a minimal container of V and suppose, in order to obtain a contradiction, that T is not irreducible container. Then there exists $T' \in \mathsf{PCov}$ such that $V \subseteq T' \subset T$. Hence, there exists a container of V such that $|T|' < |T|$, which contradicts the fact that T is minimal container. ■

Note, however, that T is an irreducible container of V does not necessarily imply that T is a minimal container of V.

Example 7.2.1 *Let $X = \{1, 2, 3, 4\}$ and let $\mathcal{C} = \{C_1, C_2, C_3\}$, where $C_1 = \{1\}$, $C_2 = \{3, 4\}$ and $C_3 = \{1, 2, 4\}$. Then $\mathsf{PCov} = \{\{1\}, \{3, 4\}, \{1, 2, 4\}, \{1, 3, 4\}, \{1, 2, 3, 4\}\}$. Let $V = \{4\}$, then irreducible containers of V are $\{3, 4\}$ and $\{1, 2, 4\}$, but $\{1, 2, 4\}$ is not a minimal container of V.*

We intend to define a number of problems associated with irreducible container, and establish the complexity results for those problems. Most interestingly, we would

[1] An equivalent definition is that there does not exist $T' \in \mathsf{PCov}$ such that $V \subseteq T' \subset T$.

like to explore any problems in the context of RBAC and other access control models that are related to the irreducible container problems.

On the other hand, we are interesting in extending the minimal container problem to a weighted minimal container problem. One obvious formulation is to associate a weight with each element of X, and the weighted minimal container problem then seeks to minimize the total weight of elements outside V. In this case, the minimal container problem is a subcase of the weighted minimal container problem where the weight of each element of X is constant. Hence the weighted minimal container problem is **NP**-hard. We would also like to examine the applications of the weighted minimal container problem and design heuristic algorithms for this problem based on the idea of our heuristic algorithms for the minimal cover problem.

References

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

[2] American National Standards Institute. *American National Standard for Information Technology – Role Based Access Control*, 2004. ANSI INCITS 359-2004.

[3] R. J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley and Sons, 2nd edition, 2008.

[4] C. A. Ardagna, M. Cremonini, E. Damiani, S. De Capitani di Vimercati, and P. Samarati. Supporting location-based conditions in access control policies. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, pages 212–222, 2006.

[5] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag, 1999.

[6] R. Awischus. Role based access control with the security administration manager (SAM). In *Proceedings of the Second ACM Workshop on Role-Based Access Control*, pages 61–68, 1997.

[7] E. Barka and R. S. Sandhu. Framework for role-based delegation models. In *Proceedings of the 16th Annual Computer Security Applications Conference*, pages 168–176, 2000.

[8] D. E. Bell. Secure computer systems: A refinement of the mathematical model. Technical Report MTR-2547, Volume III, Mitre Corporation, 1973.

[9] D. E. Bell. Secure computer systems: A network interpretation. In *Proceedings of the Third Annual Computer Security Application Conference*, pages 32–39, 1986.

[10] D. E. Bell and L. LaPadula. Secure computer systems: A mathematical model. Technical Report MTR-2547, Volume II, Mitre Corporation, 1973.

[11] D. E. Bell and L. LaPadula. Secure computer systems: Mathematical foundations. Technical Report MTR-2547, Volume I, Mitre Corporation, 1973.

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

[13] E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security*, 4(3):191–233, 2001.

[14] E. Bertino, B. Catania, M. L. Damiani, and P. Perlasca. GEO-RBAC: A spatially aware RBAC. In *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies*, pages 29–37, 2005.

[15] 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.

[16] K. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153, Mitre Corporation, 1977.

[17] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, 2002.

[18] D. F. Brewer and M. J. Nash. The Chinese wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 206–214, 1989.

[19] H. Chen and N. Li. Constraint generation for separation of duty. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies*, pages 130–138, 2006.

[20] L. Chen and J. Crampton. Applications of the oriented permission role-based access control model. In *Proceedings of the 26th IEEE International Performance, Computing, and Communications Conference*, pages 387–394, 2007.

[21] L. Chen and J. Crampton. Inter-domain role mapping and least privilege. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, pages 157–162, 2007.

[22] L. Chen and J. Crampton. On spatio-temporal constraints and inheritance in role-based access control. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*, pages 356–369, 2008.

[23] L. Chen and J. Crampton. Set covering problems in role-based access control. In *Proceedings of the 14th European Symposium on Research in Computer Security*, pages 689–704, 2009.

[24] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[25] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 184–194, 1987.

[26] Commission of the European Communities. *Information Technology Security Evaluation Criteria*, 1991. Version 1.2.

[27] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[28] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd. Securing context-aware applications using environment roles. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, pages 10–20, 2001.

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

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

[31] J. Crampton. Understanding and developing role-based administrative models. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, pages 158–167, 2005.

[32] J. Crampton. Why we should take a second look at access control in Unix. In *Proceedings of the 13th Nordic Workshop on Secure IT Systems*, pages 27–38, 2008.

[33] J. Crampton and H. Khambhammettu. Delegation in role-based access control. *International Journal of Information Security*, 7(2):123–136, 2008.

[34] S. De Capitani di Vimercati, P. Samarati, and S. Jajodia. Policies, models, and languages for access control. In *Proceedings of the Fourth International Workshop on Databases in Networked Information Systems*, pages 225–237, 2005.

[35] D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

[36] S. Du and J. B. D. Joshi. Supporting authorization query and inter-domain role mapping in presence of hybrid role hierarchy. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies*, pages 228–236, 2006.

[37] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[38] D. F. Ferraiolo, J. Cugini, and D. R. Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of the 11th Annual Computer Security Application Conference*, pages 241–248, 1995.

[39] D. F. Ferraiolo, D. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Artech House, 2nd edition, 2007.

[40] D. F. Ferraiolo and D. R. Kuhn. Role-based access controls. In *Proceedings of the 15th National Computer Security Conference*, pages 554–563, 1992.

[41] D. F. Ferraiolo, D. R. Kuhn, and R. S. Sandhu. RBAC standard rationale: Comments on "A critique of the ANSI standard on role-based access control". *IEEE Security & Privacy*, 5(6):51–53, 2007.

[42] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[43] C. K. Georgiadis, I. Mavridis, G. Pangalos, and R. K. Thomas. Flexible team-based access control using contexts. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, pages 21–27, 2001.

[44] V. D. Gligor. Characteristics of role-based access control. In *Proceedings of the First ACM Workshop on Role-Based Access Control*, page 10, 1995.

[45] V. D. Gligor, S. I. Gavrila, and D. F. Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 172–183, 1998.

[46] D. Gollmann. *Computer Security*. John Wiley and Sons, 2nd edition, 2005.

[47] F. C. Gomes, C. N. Meneses, P. M. Pardalos, and G. V. R. Viana. Experimental analysis of approximation algorithms for the vertex cover and set covering problems. *Computers & Operations Research*, 33(12):3520–3534, 2006.

[48] G. S. Graham and P. J. Denning. Protection: Principles and practice. In *Proceedings of the AFIPS Joint Computer Conferences*, pages 417–429, 1971.

[49] F. Hansen and V. Oleshchuk. SRBAC: A spatial role-based access control model for mobile systems. In *Proceedings of the Seventh Nordic Workshop on Secure IT systems*, pages 129–141, 2003.

[50] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[51] ISO. *Information technology – Open Systems Interconnection – Security frameworks for open systems: Access control framework*, 1996. ISO/IEC 10181-3.

[52] T. Jaeger and J. Tidswell. Practical safety in flexible access control models. *ACM Transactions on Information and System Security*, 4(2):158–190, 2001.

[53] D. S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, pages 38–49, 1973.

[54] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.

[55] J. B. D. Joshi and E. Bertino. Fine-grained role-based delegation in presence of the hybrid role hierarchy. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies*, pages 81–90, 2006.

[56] J. B. D. Joshi, E. Bertino, and A. Ghafoor. Temporal hierarchies and inheritance semantics for GTRBAC. In *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, pages 74–83, 2002.

[57] J. B. D. Joshi, E. Bertino, A. Ghafoor, and Y. Zhang. Formal foundations for hybrid hierarchies in GTRBAC. *ACM Transactions on Information and System Security*, 10(4), 2008.

[58] 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.

[59] G. Karjoth. Access control with IBM Tivoli access manager. *ACM Transactions on Information and System Security*, 6(2):232–257, 2003.

[60] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[61] M. Kuhlmann, D. Shohat, and G. Schimpf. Role mining - revealing business roles for security administration using data mining technology. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, pages 179–186, 2003.

[62] 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.

[63] B. W. Lampson. Protection. In *Proceedings of the Fifth Princeton Symposium on Information Sciences and Systems*, pages 437–443, 1971.

[64] N. Li, J.-W. Byun, and E. Bertino. A critique of the ANSI standard on role based access control. *IEEE Security & Privacy*, 5(6):41–49, 2007.

[65] N. Li, M. V. Tripunitara, and Z. Bizri. On mutually exclusive roles and separation-of-duty. *ACM Transactions on Information and System Security*, 10(2), 2007.

[66] J. McLean. Reasoning about security models. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 123–133, 1987.

[67] J. D. Moffett and E. C. Lupu. The uses of role hierarchies in access control. In *Proceedings of the Fourth ACM Workshop on Role-Based Access Control*, pages 153–160, 1999.

[68] M. Niezette and J. Stevenne. An efficient symbolic representation of periodic time. In *Proceedings of the First International Conference on Information and Knowledge Management*, 1992.

[69] 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*, pages 129–144, 1995.

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

[71] OASIS. *eXtensible Access Control Markup Language (XACML) Version 2.0*, 1 February 2005. OASIS Standard (T. Moses, editor).

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

[73] S. L. Osborn, R. S. 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.

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

[75] I. Ray and M. Kumar. Towards a location-based mandatory access control model. *Computers & Security*, 25(1):36–44, 2006.

[76] I. Ray and M. Toahchoodee. A spatio-temporal role-based access control model. In *Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 211–226, 2007.

[77] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceeding of the IEEE*, 63(9):1278–1308, 1975.

[78] P. Samarati and S. De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In *Proceedings of the Foundations of Security Analysis and Design, Tutorial Lectures*, pages 137–196, 2000.

[79] R. S. Sandhu. The typed access matrix model. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 122–136, 1992.

[80] R. S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, 1993.

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

[82] R. S. Sandhu. Role activation hierarchies. In *Proceedings of the Third ACM Workshop on Role-Based Access Control*, pages 33–40, 1998.

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

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

[85] M. Sipser. *Introduction to the Theory of Computation*. Course Technology, 2nd edition, 2005.

[86] M. Strembeck and G. Neumann. An integrated approach to engineer and enforce context constraints in RBAC environments. *ACM Transactions on Information and System Security*, 7(3):392–427, 2004.

[87] R. K. Thomas. Team-based access control (TMAC): A primitive for applying role-based access controls in collaborative environments. In *Proceedings of the Second ACM Workshop on Role-Based Access Control*, pages 13–19, 1997.

[88] N. Tuval and E. Gudes. Resolving information flow conflicts in RBAC systems. In *Proceedings of the 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 148–162, 2006.

[89] USA Department of Defense. *Department of Defense Trusted Computer Systems Evaluation Criteria*, 1985. DoD 5200.28-STD.

[90] J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: Finding a minimal descriptive set. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, pages 175–184, 2007.

[91] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[92] H. Wang and S. L. Osborn. Delegation in the role graph model. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies*, pages 91–100, 2006.

[93] H. Wang and S. L. Osborn. Discretionary access control with the administrative role graph model. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, pages 151–156, 2007.

[94] G. T. Wickramaarachchi, W. H. Qardaji, and N. Li. An efficient framework for user authorization queries in RBAC systems. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, pages 23–32, 2009.

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

[96] Y. Zhang and J. B. D. Joshi. UAQ: A framework for user authorization query processing in RBAC extended with hybrid hierarchy and constraints. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, pages 83–92, 2008.