CERIAS Tech Report 2009-10
ACCESS CONTROL POLICY MANAGEMENT
by Qihua Wang
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

ACCESS CONTROL POLICY MANAGEMENT

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Qihua Wang

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2009

Purdue University

West Lafayette, Indiana

To my father and my mother.

ACKNOWLEDGMENTS

Throughout my graduate study at Purdue University, I have been very fortunate to meet with great people, who have had a dramatic impact on my research and my accomplishments.

I am very grateful to my advisor Professor Ninghui Li, who is always patient and available to students. He has provided me a lot of support, encouragement, and freedom during my PhD study. His insights and suggestions have been invaluable to my work. I am also very grateful to Professor Elisa Bertino, Dr. Jorge Lobo from IBM Watson Research Center, and Dr. Hongxia Jin from IBM Almaden Research Center. I have greatly benefited from the close collaboration with them. Finally, I am very thankful to my other research collaborators and my fellow students. They have provided so much help on my research and other aspects of life during my graduate study.

My graduate study was conducted in the Center for Education and Research in Information Assurance and Security (CERIAS). I benefited a lot from the research and education environment that CERIAS provided. I thank all the faculties, staffs, and colleagues in CERIAS for their support.

Last but not least, I would like to thank my parents for their unconditional love, encouragement, and support.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

FPT	Fixed-Parameter Tractable
NP	Non-deterministic Polynomial Time
P	Polynomial Time
QSAT	Quantified Satisfiability Problem
RBAC	Role-Based Access Control
R^2BAC	Role-and-Relation-Based Access Control
SAT	Boolean Satisfiability Problem
SMER	Static Mutually Exclusive Roles
SoD	Separation of Duty
SSoD	Static Separation of Duty

ABSTRACT

Wang, Qihua. Ph.D., Purdue University, May 2009. Access Control Policy Management. Major Professor: Ninghui Li.

Access control is the traditional center of gravity of computer security [1]. People specify access control policies to control accesses to resources in computer systems. The management of access control policies include policy specification and policy analysis. In this dissertation, we design a new language for policy specification, propose a new type of access control policy, and study the computational complexity of a variety of policy analysis problems. In particular,

- We design a novel algebra that enables the specification of high-level security policies that combine qualification requirements with quantity requirements. Our algebra contains six operators and is expressive enough to specify many natural high-level security policies. We study the properties of the algebra, as well as several computational problems related to the algebra.

- Traditional access control policy analysis focuses on restricting access. However, an equally important aspect of access control is to enable access. With this in mind, we introduce the notion of resiliency policies for access control systems. We formally define resiliency policies and study computational problems on checking whether an access control state satisfies a resiliency policy. We also study the consistency between resiliency policies and separation of duty policies.

- The workflow authorization system is a popular access control model. We study fundamental problems related to policy analysis in workflow authorization systems, such as determining whether a set of users can complete a workflow in a certain access control state. In particular, we apply tools from parameterized complexity

theory to better understand the complexities of such problems. We also introduce the notion of resiliency to workflow authorization systems.

- Delegation is an important tool to provide flexibility and enforce resiliency in access control systems. However, delegation may also allow colluding users to bypass security policies. We study the security impact of delegation and formally define the notion of security with regard to delegation. We propose mechanisms to enforce delegation security. In particular, we design a novel source-based enforcement mechanism for workflow authorization systems so as to achieve both security and efficiency. Finally, we discuss how to use delegation to meet resiliency requirements.

1 INTRODUCTION

Access control is the traditional center of gravity of computer security [1]. People specify access control policies to control accesses to resources in computer systems. A high-level access control policy states an overall requirement for a sensitive task. A well-known high-level access control policy is Separation of Duty (SoD), which is widely recognized as a fundamental principle in computer security [2, 3]. In its simplest form, the principle states that a sensitive task should be performed by two different users acting in cooperation. The concept of SoD has long existed before the information age; it has been widely used in, for example, the banking industry and the military, sometimes under the name "the two-man rule". More generally, an SoD policy requires the cooperation of at least k different users to complete a task. SoD has been identified as a high-level mechanism that is "at the heart of fraud and error control" [2].

In many situations, however, it is not enough to require only that k different users be involved in a sensitive task; there are also minimal qualification requirements for these users. For example, one may want to require users involved in a task to be physicians, certified nurses, certified accountants, or directors of a company. It is thus desirable to introduce a concise language that enables the formal specification of high-level policies that combine requirements on users' attributes with requirements on the number of users motivated by separation of duty considerations.

Furthermore, while policy specification and analysis has been a main research area in access control for several decades, almost all existing work focuses on properties which ensure that users who should not have access do not get access. Such focus on safety properties probably stems from the fact that access control has been mostly viewed as a tool for restricting access. However, an equally important aspect of access control is to enable access (selectively). In this dissertation, we introduce the notion of resiliency

policies for access control systems, which require an access control system to be resilient to the absence of users.

Both SoD policies and resiliency policies are high-level security policies. They state an overall requirement without referring to individual steps in the task. High-level security policies are enforced by lower-level schemes such as workflow authorization systems and the delegation of users' privileges. Workflows are used in numerous domains, including production, purchase order processing, and various management tasks. A workflow divides a task into a set of well-defined sub-tasks (called *steps* here). Workflow authorization systems manage access control in workflows and have gained popularity in the research community [4–8]. Security policies in workflow authorization systems are usually specified using authorization constraints. One may specify, for each step, which users are authorized to perform it. In addition, one may specify the constraints between users who perform different steps in the workflow. For example, one may require that two steps must be performed by different users for the purpose of separation of duty [2]. For another example, one may need two steps be performed by the same user so to enforce binding of duty policies [6]. As we can see in the two examples, equality and inequality are two binary relations widely used in constraints of workflow authorization systems. In this dissertation, we introduce and study more complex workflow constraints that support user-defined binary relations, such as "be supervisor of" and "no conflict of interests".

We have discussed workflow authorization system as a mechanism to enforce high-level security policies, such as SoD policies. To enforce resiliency policies, one may introduce enough redundancy of human resources in the system configuration. An alternative approach is to use user-to-user delegation (or delegation for short). Delegation is a mechanism that allows a user A to act on another user B's behalf by making B's access rights available to A. It is well recognized as an important mechanism to provide fault-tolerance and flexibility in access control systems, and has gained popularity in the research community [9–17].

Essentially, a delegation operation temporarily changes the access control state so as to allow a user to use another user's access privileges. While delegation can make an

access control system more resilient to the absence of users, it may lead to violation of security policies, especially static separation of duty policies. For instance, if role r_1 and role r_2 are mutually exclusive, then a user who is a member of r_1 should not be allowed to receive r_2 from others through delegation. In contrast to normal access right administration operations, which are performed centrally, delegation operations are usually performed in a distributed manner. That is to say, users have certain control on the delegation of their own rights. As we will see in Section 5.1.1, delegation may introduce security breaches into an access control system, which allow colluding users to circumvent security policies. Due to the decentralized nature of delegation and the fact that not all the users in the system are trusted, collusion is a threat that must not be overlooked. In this dissertation, we study the security impact of delegation on access control systems in detail.

Thesis Statement

The goal of this dissertation is to improve the state-of-the-art of access control policy management. More specifically, our goal is to design new types of access control policies that are useful in practice, propose formal languages to specify such policies, and study effective mechanisms to enforce such policies.

Contributions

The contributions of this dissertation are summarized as follows.

- We propose a novel algebra that enables the specification of high-level security policies that combine qualification requirements with quantity requirements. Our algebra contains six operators and is expressive enough to specify many natural high-level security policies. For example, the term $(\texttt{Accountant} \sqcup \texttt{Treasurer})^{+}$ requires that all participants must be either an `Accountant` or a `Treasurer`; while the term $((\texttt{Physician} \sqcup \texttt{Nurse}) \otimes (\texttt{Manager} \wedge \neg\texttt{Accountant}))$ requires two different users, one of who is either a `Physician` or a `Nurse`, and the other is a `Manager` but not

an `Accountant`. We study the algebraic properties of the algebra, as well as several computational problems related to the algebra.

- We formally define resiliency policies, which require an access control system to be resilient to the absence of users. In its general form, a resiliency policy states that upon removal of any s users, there should still exist d disjoint sets of users such that the users in each set together possess certain permissions of interest. We study computational problems on checking whether an access control state satisfies a resiliency policy. We study the consistency between resiliency policies and separation of duty policies.

- We propose the role-and-relation-based access control (R^2BAC) model for workflow authorization systems. In R^2BAC, in addition to a user's role memberships, the user's relationships with other users help determine whether the user is allowed to perform a certain step in a workflow. For example, a constraint may require that two steps must not be performed by users who have conflicts of interests. R^2BAC is a natural step beyond Role-Based Access Control (RBAC) [18], especially in the setting of workflows. As a role defines a set of users, which can be viewed as a unary relation among the set of all users, a binary relation is the natural next step.

- We study fundamental problems in workflow authorization systems, such as determining whether a set of users can complete a workflow and checking whether a workflow is resilient to the absence of users. In particular, we apply tools from parameterized complexity theory to better understand the complexities of some of these problems.

- We study the impact of delegation on the security of workflow authorization systems. We formally define the notion of security with respect to delegation and propose mechanisms to enforce delegation security in workflow authorization systems. We also discuss how to use delegation to meet resiliency requirements.

Organization of the Dissertation

This dissertation is organized as follows. In Chapter 2, we propose an algebra for high-level security policy specification. In Chapter 3, we introduce the notion of resiliency policy for access control systems. We then study the satisfiability and resiliency problems in workflow authorization systems in Chapter 4. After that, we study delegation, which is an important mechanism to enforce delegation, in Chapter 5. Finally, we discuss related work in Chapter 6 and conclude in Chapter 7.

2 AN ALGEBRA FOR HIGH-LEVEL ACCESS CONTROL POLICY SPECIFICATION

As the focus of the first step of secure task design, a high-level access control policy (or equivalently, a high-level security policy) states an overall requirement that must be satisfied by any set of users that together complete a task. As stated in Chapter 1, a well-known high-level security policy is Separation of Duty (SoD), which requires the cooperation of at least k different users to complete a task.

In many situations, however, it is not enough to require only that k different users be involved in a sensitive task; there are also minimal qualification requirements for these users. For example, one may want to require users involved in a task to be physicians, certified nurses, certified accountants, or directors of a company. Partly due to the lack of a concise-yet-expressive language for specifying such high-level security policies, people usually skip the formal specification of high-level security policies (perhaps expressing high-level security policies in a natural language) and specify qualification requirements at the level of enforcement mechanism. For example, if a designer believes that a task should involve a manager and two clerks, she may create a workflow with three steps and require two clerks to each perform Step 1 and Step 3, and a manager to perform Step 2.

However, formal specification of high-level security policies provides a number of important advantages. First of all, formal specification minimizes the possibility of misunderstanding between policy designers and system designers. Using a natural language could lead to ambiguity and misinterpretation, and are thus inappropriate to specify security policies, as a flaw in a policy could lead to major security breaches. Second, formal specification facilitates the analysis of security policies. Given a formal policy specification language, we may develop tools to analyze formally-specified policies, such as checking whether certain groups of users satisfy a policy, so as to detect policies that are too restrictive or too permissive when compared to actual needs in practice. It is beneficial to detect

design flaws at an early stage, because low-level enforcement schemes, which contain execution details of tasks, are usually more difficult to analyze than high-level policies. As we will see in Example 2 in Section 5.2.2, low-level enforcement schemes such as workflows with security constraints may involve other factors in addition to security requirements, which complicates the analysis on those enforcement schemes. Finally, formal specification of high-level policies allows us to develop tools to verify whether a low-level enforcement scheme is compliant with a high-level security policy. For example, a workflow may contain branches and loops; it is important to verify that no route in the workflow bypasses the high-level security policy. As manual verification is time-consuming and error-prone, formal verification tools are highly desirable.

In this chapter, we introduce a novel algebra that enables the formal specification of high-level policies that combine qualification requirements with quantity requirements motivated by separation of duty considerations. A term in our algebra specifies a requirement on sets of users (we call these *usersets*). A high-level policy, which associates a task with a term in the algebra, requires that all sets of users that complete an instance of the task satisfy the term. Our algebra has four binary operators: $\sqcup, \sqcap, \odot, \otimes$, and two unary operators $\neg, +$. An SoD policy that requires 3 different users can be expressed using the term $(\mathsf{All} \otimes \mathsf{All} \otimes \mathsf{All})$, where All is a keyword that refers to the set of all users. A policy that requires either a manager or two different clerks is expressed using the term $(\mathtt{Manager} \sqcup (\mathtt{Clerk} \otimes \mathtt{Clerk}))$.

The remainder of this chapter is organized as follows. We introduce the syntax and semantics of the algebra in Section 2.1. We then discuss different enforcement mechanisms for policies specified in the algebra in Section 2.2. In Sections 2.3, 2.4, and 2.5, we study computational problems related to analysis and enforcement of policies. In Section 2.6, we discuss extensions to the syntax of the algebra, the relationship between the algebra and regular expressions, as well as limitations of the expressive power of the algebra. Proofs not included in the main body are included in the appendices unless otherwise stated.

2.1 The Algebra

In this section, we introduce an algebra for expressing high-level security policies.

2.1.1 Syntax, Semantics, and Examples

In our definition of the algebra, we use the notion of roles. We use a role to denote a set of users that have some common qualification or common job responsibility. We emphasize, however, that the algebra is not restricted to Role-Based Access Control (RBAC) systems [18]. In our algebra, a role is simply a named set of users. The notion of roles can be replaced by groups or user attributes. We use $\mathcal{U}$ to denote the set of all users, and $\mathcal{R}$ to denote the set of all roles.

Definition 2.1.1 (Terms in the Algebra) Terms in the algebra are defined as follows:

- An *atomic term* takes one of the following three forms: a role $r \in \mathcal{R}$, the keyword All, or a set $S \subseteq \mathcal{U}$ of users.
- An atomic term is a *unit term*; furthermore, if ϕ_1 and ϕ_2 are unit terms, then $\neg\phi_1$, $(\phi_1 \sqcap \phi_2)$ and $(\phi_1 \sqcup \phi_2)$ are also unit terms.
- A unit term is a *term*; if ϕ is a unit term, then ϕ^+ is a term; if ϕ_1 and ϕ_2 are terms, then $(\phi_1 \sqcup \phi_2)$, $(\phi_1 \sqcap \phi_2)$, $(\phi_1 \otimes \phi_2)$, and $(\phi_1 \odot \phi_2)$ are also terms.

The unary operator $\neg$ has the highest priority, followed by the unary operator $+$, then by the four binary operators (namely $\sqcap$, $\sqcup$, $\odot$, $\otimes$), which have the same priority.

We now give several simple example terms to illustrate the intuition behind the operators in the algebra. The term "$(\texttt{Manager} \sqcap \texttt{Accountant})$" requires a user that is both a `Manager` and an `Accountant`. The term "$(\texttt{Manager} \sqcap \neg\{Alice, Bob\})$" requires a user that is a manager, but is neither Alice nor Bob; here, the sub-term "$\neg\{Alice, Bob\}$" implements a blacklist. The term "$(\texttt{Physician} \sqcup \texttt{Nurse})$" requires a user that is either a `Physician` or a `Nurse`. The term "$(\texttt{Manager} \odot \texttt{Clerk})$" requires a user who is a `Manager` and a user

who is a `Clerk`; in particular, when one user is both a `Manager` and a `Clerk`, that user by himself also satisfies the requirement. The term "$((\mathsf{All} \otimes \mathsf{All}) \otimes \mathsf{All})$" requires three different users. The keyword All allows us to refer to the set of all users. The term "$\texttt{Accountant}^+$" requires a set of one or more users, where each user in the set is an `Accountant`.

To formally assign meanings to terms, we need to first assign meanings to the roles used in the term. For this, we introduce the notion of configurations.

Definition 2.1.2 (Configurations) A *configuration* is given by a pair $\langle U, UR \rangle$, where U denotes the set of all users in the configuration, and $UR \subseteq U \times \mathcal{R}$ determines role memberships. When $(u, r) \in UR$, we say that u is a member of the role r.

Note that in a configuration $\langle U, UR \rangle$, UR should not be confused with the user-role assignment relation UA in RBAC. When an RBAC system has both UA and a role hierarchy RH, the two relations UA and RH together determine UR.

When describing the UR relation, we often use U_r to denote the set of users assigned to role r, i.e. $U_r = \{u \mid (u, r) \in UR\}$.

Definition 2.1.3 (Satisfaction of a Term) Given a configuration $\langle U, UR \rangle$, we say that a userset $X \subseteq U$ *satisfies* a term ϕ under $\langle U, UR \rangle$ if and only if one of the following holds[1]:

- The term ϕ is the keyword All, and X is a singleton set $\{u\}$ such that $u \in U$.
- The term ϕ is a role r, and X is a singleton set $\{u\}$ such that $(u, r) \in UR$.
- The term ϕ is a set S of users, and X is a singleton set $\{u\}$ such that $u \in S$.
- The term ϕ is of the form $\neg\phi_0$ where ϕ_0 is a unit term, and X is a singleton set that does not satisfy ϕ_0.
- The term ϕ is of the form ϕ_0^+ where ϕ_0 is a unit term, and X is a nonempty userset such that for every $u \in X$, $\{u\}$ satisfies ϕ_0.
- The term ϕ is of the form $(\phi_1 \sqcup \phi_2)$, and either X satisfies ϕ_1 or X satisfies ϕ_2.

[1] We sometimes say X satisfies ϕ, and omit "under $\langle U, UR \rangle$" when the configuration is clear from the context.

- The term ϕ is of the form $(\phi_1 \sqcap \phi_2)$, and X satisfies both ϕ_1 and ϕ_2.
- The term ϕ is of the form $(\phi_1 \otimes \phi_2)$, and there exist usersets X_1 and X_2 such that $X_1 \cup X_2 = X$, $X_1 \cap X_2 = \emptyset$, X_1 satisfies ϕ_1, and X_2 satisfies ϕ_2.
- The term ϕ is of the form $(\phi_1 \odot \phi_2)$, and there exist usersets X_1 and X_2 such that $X_1 \cup X_2 = X$, X_1 satisfies ϕ_1, and X_2 satisfies ϕ_2. This differs from the definition for $\otimes$ in that it does not require $X_1 \cap X_2 = \emptyset$.

For example, given the term $(\texttt{Manager} \odot \texttt{Clerk})$, and the configuration $\langle U = \{Alice, Bob\},\ UR\rangle$, in which UR is such that: $U_{\texttt{Manager}} = \{Alice\}$ and $U_{\texttt{Clerk}} = \{Alice, Bob\}$, we have $\{Alice\}$ satisfies the term and $\{Alice$,$Bob\ \}$ also satisfies the term.

Intuitively, a configuration $\langle U, UR\rangle$ represents the access control state of an organizational unit, a term ϕ defines the security requirement of a sensitive task T, and $X \subseteq U$ is a set of users in the organizational unit who are about to perform T. X satisfying ϕ indicates that the set of users meet the security requirement of T. Also, it is clear from Definition 2.1.3 that no term can be satisfied by an empty set.

The following examples help illustrate that one can express sophisticated policies in the algebra.

- $\{Alice, Bob, Carl\} \otimes \{Alice, Bob, Carl\}$

  This term is satisfied by any two users out of the list of three.

- $(\texttt{Accountant} \sqcup \texttt{Treasurer})^+$

  This term requires that all participants must be either an `Accountant` or a `Treasurer`. But there is no restriction on the number of participants except that the number is non-zero.

- $((\texttt{Manager} \odot \texttt{Accountant}) \otimes \texttt{Treasurer})$

  This term is satisfied by a userset consisting of a `Manager`, an `Accountant`, and a `Treasurer`; the first two requirements can be satisfied by a single user.

- $((\texttt{Physician} \sqcup \texttt{Nurse}) \otimes (\texttt{Manager} \sqcap \neg\texttt{Accountant}))$

  This term is satisfied by a userset consisting of two different users, one of who is either a `Physician` or a `Nurse`, and the other is a `Manager`, but not an `Accountant`.

- $((\texttt{Manager} \odot \texttt{Accountant} \odot \texttt{Treasurer}) \sqcap (\texttt{Clerk} \sqcap \neg\{Alice, Bob\})^{+})$

  This term is satisfied by a userset consisting of a `Manager`, an `Accountant` and a `Treasurer`. In addition, everybody in the userset must be a `Clerk` and must not be $Alice$ or Bob.

2.1.2 Satisfaction Trees

When a userset X satisfies a term ϕ under a configuration $\langle U, UR\rangle$, some subterms of ϕ are satisfied by subsets of X. We formalize this by the notion of a satisfaction tree. A satisfaction tree serves as an evidence of X satisfying ϕ that can be easily verified.

Definition 2.1.4 (Satisfaction Tree) Given a term ϕ and a configuration $\langle U, UR\rangle$, we say that T is a satisfaction tree of ϕ under $\langle U, UR\rangle$ if and only if the following three conditions hold.

1. T is a syntax tree of ϕ, where each inner node of T denotes a binary operator in ϕ, and each leaf node denotes a sub-term of ϕ that is either a unit term or takes the form ϕ_0^+. That is, sub-terms of the form ϕ_0^+ are not not further decomposed in T and are represented as leaves.

2. Each node N in T is labeled with a (possibly empty) set of users, which is denoted as $L_T(N)$, and the following rules hold for every node N in T. We denote N_1 and N_2 as the left and right children of N, respectively.

   - When N is a leaf node representing a unit term ϕ_0: either $L_T(N) = \emptyset$ or $L_T(N) = \{u\}$ satisfies ϕ_0 under $\langle U, UR\rangle$.
   - When N is a leaf node representing a sub-term ϕ_0^+: either $L_T(N) = \emptyset$ or $L_T(N) = X$ satisfies ϕ_0^+ under $\langle U, UR\rangle$.

- When N represents $\sqcup$: either $(L_T(N) = L_T(N_1) \wedge L_T(N_2) = \emptyset)$ or $(L_T(N) = L_T(N_2) \wedge L_T(N_1) = \emptyset)$.
- When N represents $\sqcap$: $L_T(N) = L_T(N_1) = L_T(N_2)$.
- When N represents $\odot$: $L_T(N) = L_T(N_1) \cup L_T(N_2)$, and $(L_T(N) \neq \emptyset) \Rightarrow (L_T(N_1) \neq \emptyset \wedge L_T(N_2) \neq \emptyset)$, where $\Rightarrow$ denote logic implication.
- When N represents $\otimes$: $L_T(N) = L_T(N_1) \cup L_T(N_2)$, $L_T(N_1) \cap L_T(N_2) = \emptyset$, and $(L_T(N) \neq \emptyset) \Rightarrow (L_T(N_1) \neq \emptyset \wedge L_T(N_2) \neq \emptyset)$.

3. $L_T(N_r) \neq \emptyset$, where N_r is the root of the tree T.

According to the conditions in the above definition, it can be easily shown that $L_T(N) \subseteq L_T(N')$, when N' is an ancestor of N in the satisfaction tree T.

Intuitively, in a satisfaction tree T, a node is either labeled with a userset that satisfies the sub-term represented by the sub-tree of T rooted at the node, or labeled with $\emptyset$, indicating that the node is in a branch connected by $\sqcup$ and the sub-term represented by that branch does not need to be satisfied (because the other branch is satisfied). The following lemma formalizes this intuition.

Lemma 2.1.1 Let T be a satisfaction tree of ϕ under $\langle U, UR\rangle$, for each node N in T, if $L_T(N) \neq \emptyset$, then $L_T(N)$ satisfies the sub-term of ϕ represented by the sub-tree of T rooted at N.

The following theorem relates the existence of a satisfaction tree for a term ϕ with the satisfiability of ϕ.

Theorem 2.1.2 Given a configuration $\langle U, UR\rangle$ and a term ϕ, a userset X satisfies ϕ under $\langle U, UR\rangle$ if and only if there exists a satisfaction tree of ϕ such that $L_T(N_r) = X$, where N_r is the root of T.

The proofs of Lemma 2.1.1 and Theorem 2.1.2 are given in Appendix A.1.

2.1.3 Evaluating a Term to a Set of Usersets

Given a configuration $\langle U, UR\rangle$ and a term ϕ, ϕ may be satisfied by multiple usersets, thus we can say that ϕ evaluates to a set of usersets.

Definition 2.1.5 (Value of a Term) Given a configuration $\langle U, UR\rangle$ and a term ϕ, $S_{\langle U,UR\rangle}(\phi)$ denotes the set of all usersets that satisfy ϕ under $\langle U, UR\rangle$, and is called the *value* of term ϕ under the configuration.

Example 1 Consider the term $\phi = ((\texttt{Manager} \odot \texttt{Accountant} \odot \texttt{Treasurer}) \sqcap (\texttt{Clerk} \sqcap \neg\{Alice, Bob\})^+)$ and the configuration $\langle U = \{Alice, Bob, Carl, Doris, Elaine, Frank\},\ UR\rangle$, in which UR is such that:

$$\begin{aligned} U_{\texttt{Manager}} &= \{Alice, Doris, Elaine\} \\ U_{\texttt{Accountant}} &= \{Doris, Frank\} \\ U_{\texttt{Treasurer}} &= \{Bob, Carl, Doris\} \\ U_{\texttt{Clerk}} &= \{Alice, Bob, Carl, Doris, Frank\}. \end{aligned}$$

The sub-term $(\texttt{Clerk} \sqcap \neg\{Alice, Bob\})^+$ blacklists $Alice$ and Bob so that only subsets of $\{Carl, Doris, Frank\}$ may satisfy ϕ. We have

$$S_{\langle U,UR\rangle}(\phi) = \{\ \{Doris\},\ \{Carl, Doris\},\ \{Doris, Frank\}, \{Carl, Doris, Frank\}\ \}$$

That is, there are four usersets that satisfy the term ϕ.

2.1.4 Algebraic Properties

We now introduce the notion of equivalence among terms, which enables us to study the algebraic properties of the operators in the algebra.

Definition 2.1.6 (Term Equivalence) We say that two terms ϕ_1 and ϕ_2 are *equivalent* (denoted by $\phi_1 \equiv \phi_2$) when for every userset X and every configuration $\langle U, UR\rangle$, X satisfies ϕ_1 under $\langle U, UR\rangle$ if and only if X satisfies ϕ_2 under $\langle U, UR\rangle$. In other words, $\phi_1 \equiv \phi_2$ if and only if $\forall\langle U, UR\rangle\ \left[S_{\langle U,UR\rangle}(\phi_1) = S_{\langle U,UR\rangle}(\phi_2)\right]$.

Using a straightforward induction on the structure of terms, one can show that if $\phi_1 \equiv \phi_2$, then, for any term ϕ in which ϕ_1 occurs, let ϕ' be the term obtained by replacing in ϕ one or more occurrences of ϕ_1 with ϕ_2, we have $\phi \equiv \phi'$.

Theorem 2.1.3 The operators have the following algebraic properties:

1. The operators $\sqcup, \sqcap, \odot, \otimes$ are commutative and associative. That is, for each $\mathsf{op} \in \{\sqcup, \sqcap, \odot, \otimes\}$, and any terms ϕ_1, ϕ_2, and ϕ_3, we have $(\phi_1 \mathsf{\ op\ } \phi_2) \equiv (\phi_2 \mathsf{\ op\ } \phi_1)$ and $((\phi_1 \mathsf{\ op\ } \phi_2) \mathsf{\ op\ } \phi_3) \equiv (\phi_1 \mathsf{\ op\ } (\phi_2 \mathsf{\ op\ } \phi_3))$.

2. The operators $\sqcup$ and $\sqcap$ distribute over each other. That is, $(\phi_1 \sqcup (\phi_2 \sqcap \phi_3)) \equiv ((\phi_1 \sqcup \phi_2) \sqcap (\phi_1 \sqcup \phi_3))$ and $(\phi_1 \sqcap (\phi_2 \sqcup \phi_3)) \equiv ((\phi_1 \sqcap \phi_2) \sqcup (\phi_1 \sqcap \phi_3))$.

3. The operator $\odot$ distributes over $\sqcup$. That is, $(\phi_1 \odot (\phi_2 \sqcup \phi_3)) \equiv ((\phi_1 \odot \phi_2) \sqcup (\phi_1 \odot \phi_3))$.

4. The operator $\otimes$ distributes over $\sqcup$. That is, $(\phi_1 \otimes (\phi_2 \sqcup \phi_3)) \equiv ((\phi_1 \otimes \phi_2) \sqcup (\phi_1 \otimes \phi_3))$.

5. No other ordered pair of binary operators has the distributive property. (There are 12 such pairs altogether; the four of them listed above have the distributive property.)

6. $(\phi_1 \sqcap \phi_2)^+ \equiv (\phi_1^+ \sqcap \phi_2^+)$

7. DeMorgan's Law: $\neg(\phi_1 \sqcap \phi_2) \equiv (\neg\phi_1 \sqcup \neg\phi_2)$, $\neg(\phi_1 \sqcup \phi_2) \equiv (\neg\phi_1 \sqcap \neg\phi_2)$

See Appendix A.1 for the proof of the above theorem, which also gives a counterexample for each case that the distributive property does not hold.

Because of the associativity properties, in the rest of this chapter we omit parentheses in a term when doing so does not cause any confusion.

We now describe some other facts about the operators, to further illustrate the operators and their relationships.

- Any userset that satisfies $(\phi_1 \sqcap \phi_2)$ also satisfies $(\phi_1 \sqcup \phi_2)$, but not the other way around.

- Any userset that satisfies $(\phi_1 \sqcap \phi_2)$ also satisfies $(\phi_1 \odot \phi_2)$, but not the other way around.

- Any userset that satisfies $(\phi_1 \otimes \phi_2)$ also satisfies $(\phi_1 \odot \phi_2)$, but not the other way around.

- Any userset that satisfies $\phi_1^+ \sqcup \phi_2^+$ also satisfies $(\phi_1 \sqcup \phi_2)^+$, but not the other way around.

Proofs to the first three relationships are straightforward. Here, we prove the last one. If X satisfies $(\phi_1^+ \sqcup \phi_2^+)$, then X satisfies either ϕ_1^+ or ϕ_2^+. Without loss of generality, assume that X satisfies ϕ_1^+. Then, for every $u \in X$, $\{u\}$ satisfies ϕ_1 and thus satisfies $(\phi_1 \sqcup \phi_2)$. Hence, X satisfies $(\phi_1 \sqcup \phi_2)^+$. For the other direction, if $\{u_1\}$ satisfies ϕ_1 but not ϕ_2, and $\{u_2\}$ satisfies ϕ_2 but not ϕ_1, then $\{u_1, u_2\}$ satisfies $(\phi_1 \sqcup \phi_2)^+$ but not $\phi_1^+ \sqcup \phi_2^+$.

2.1.5 Rationale of the Design of the Algebra

We now discuss the rationale underlying some of the decisions we made in designing the algebra.

Monotonicity. SoD policies satisfy the property of monotonicity; that is, if an SoD policy requires two users to perform a task, then having three or more users certainly satisfies this policy. Similarly, one may want a security algebra like ours to also satisfy the monotonicity property; that is, if a userset X satisfies a term ϕ, then any superset of X also satisfies ϕ. McLean [19] adopts this property in his security algebra for N-person policies.

Our algebra is designed to support both monotonic policies and policies that are not monotonic. For example, the term $(\texttt{Accountant} \otimes \texttt{Accountant})$ can be satisfied only by a set of two users; a set that contains more than two users cannot satisfy the term. More generally, in Definition 2.1.3, term satisfaction is defined in such a way that every user in the userset is used to satisfy certain component of the term. No "extra" user is allowed.

We have considered a design having the monotonicity property, in which we call the notion of satisfaction in Definition 2.1.3 "strict satisfaction" and define that a userset X satisfies a term ϕ if and only if X contains a subset that strictly satisfies ϕ. We chose our current design over the one that has the monotonicity property because the current design

is more expressive. Consider the following example. When one says that "a task requires two Accountants", this may mean one of the following three policies:

1. The task must be performed by a set of two users, both of whom are Accountants. A group containing more (or less) than two people is not allowed.

2. The task must be performed by a set that contains two Accountants. In particular, a userset that contains two Accountants and a third user who is not an Accountant is allowed to perform the task.

3. The task must be performed by a set of two or more Accountants. In particular, a set of three Accountants can perform the task, but a set of two Accountants and one non-Accountant cannot. This ensures that everyone involved in the task has the qualification of an Accountant.

Policies 1 and 3 cannot be expressed using an algebra that has the monotonicity property. Suppose that one tries to use a term ϕ to express policy 1 (or policy 3) in an algebra that has the monotonicity property, then a set X of two Accountants satisfies ϕ. By monotonicity property, any superset of X also satisfies ϕ. This violates the intention of policies 1 and 3. More generally, a monotonic algebra cannot express policies that disqualify usersets that contain extra users, nor can it express security requirements in the form of "all involved users must meet certain qualification requirements".

By dropping the monotonicity property, our algebra is able to express all the three policies. Policy 1 is expressed using the term $(\texttt{Accountant} \otimes \texttt{Accountant})$. Policy 2 is expressed using the term $((\texttt{Accountant} \otimes \texttt{Accountant}) \odot \mathsf{All}^+)$. Note that the term All^+ can be satisfied by any nonempty userset. Policy 3 is expressed using the term $(\texttt{Accountant} \otimes \texttt{Accountant}^+)$.

Restrictions on "$\neg$" and "$+$". The syntax of our algebra (Definition 2.1.1) restricts that the two operators "$\neg$" and "$+$" be applied only to unit terms, i.e., those terms that do not contain $\odot$, $\otimes$, or $+$. The motivation for this design decision is the psychological acceptability principle [3]. We would like each operator to have a clear and intuitive meaning so that

when one writes down a policy as a term, there is less chance to make mistakes and one is more confident that the term expresses the intended policy.

When $\neg$ is applied to a unit term, it expresses negative qualification about a single user; this has a clear meaning; the term $\neg\phi_0$ means a user that does not satisfy ϕ_0. However, if $\neg$ is applied to a term that involves $\odot$, $\otimes$, or $+$; then the meaning becomes less clear. Consider the term $\neg(\texttt{Accountant} \odot \texttt{Manager})$. Any userset of size three does not satisfy $(\texttt{Accountant} \odot \texttt{Manager})$; therefore, it should satisfy $\neg(\texttt{Accountant} \odot \texttt{Manager})$, even if every user in the userset is both an `Accountant` and a `Manager`. It is unclear to us what kind of real-world security policies such a term expresses.

The term ϕ_0^+, when ϕ_0 is a unit term, has a clear meaning; it means that every user must satisfy ϕ_0. The same term, when ϕ_0 involves operators such as $\odot$ and $\otimes$, has at least two possible meanings. One is to interpret $+$ as the closure operator of $\odot$, that is, a userset X satisfies ϕ_0^+ if and only if X can be divided into a number of (*possibly overlapping*) subsets such that each subset satisfies ϕ_0. The other is to interpret $+$ as the closure operator for $\otimes$, that is, a userset X satisfies ϕ_0^+ if and only if X can be divided into a number of *mutually disjoint* subsets such that each subset satisfies ϕ_0. The two meanings coincide when ϕ_0 is a unit term. We could use two operators, one for each meaning, and allow them to be applied to non-unit terms. However, this adds complexity to the algebra and we have not seen a need for this. For simplicity and usability, we chose to allow $+$ only be applied to unit terms. The algebra can be extended to have two closure operators that can be applied to non-unit terms, if a need for them arises in other application domains.

2.2 Enforcing Policies Specified in the Algebra

Once a high-level security policy has been specified in the algebra, we may proceed to enforcement design step. Before doing so, it is beneficial to perform certain analyses on the high-level policy to detect design flaws at an early stage.

A basic level of sanity check is to determine whether a term is satisfiable at all, as a term that cannot be satisfied in any configuration is probably not what a policy author intended.

We define the Term Satisfiability (TSAT) problem for such an analysis. A problem similar to TSAT is the Term-Configuration Satisfiability (TCSAT) problem, which asks whether a term is satisfiable under a given configuration. This is useful when determining whether a term is meaningful in the current configuration of an organization. Formal definitions of TSAT and TCSAT are given in below, and we will study their computational complexity in Section 2.3.

Definition 2.2.1 (TSAT) Given a term ϕ, the *Term Satisfiability* (*TSAT*) *problem* determines whether there exists a configuration $\langle U, UR\rangle$ and a userset X such that X satisfies ϕ under $\langle U, UR\rangle$.

Definition 2.2.2 (TCSAT) Given a term ϕ and a configuration $\langle U, UR\rangle$, the *Term-Configuration Satisfiability* (*TCSAT*) *problem* determines whether there exists a userset X that satisfies ϕ under $\langle U, UR\rangle$.

Besides basic sanity checks on satisfiability, it is useful to select a number of targeted usersets and determine whether these usersets satisfy the term. If a set of users who are expected to perform the task guarded by the policy does not satisfy the term, the policy is too restrictive; if a set of users who should not be able to perform the task satisfies the term, the policy is too permissive. In either case, the policy is flawed and must be redesigned. We define the Userset-Term Satisfaction (UTS) problem for such an analysis. The computational complexity of UTS will be studied in Section 2.4.

Definition 2.2.3 (UTS) Given a term ϕ, a configuration $\langle U, UR\rangle$, and a userset X, the *Userset-Term Satisfaction* (*UTS*) *problem* determines whether X satisfies ϕ under $\langle U, UR\rangle$.

It is worth mentioning that UTS and TCSAT are related problems: given a configuration and a term, UTS is a decisional problem which asks whether a given userset satisfies the term, while TCSAT can be solved by searching for a userset in the configuration that satisfies the term.

If a high-level security policy passes all the tests, we need to enforce the policy correctly. A high-level security policy can be enforced statically or dynamically. In static enforcement, one ensures that in a configuration, any set of users who together have enough

permissions to perform the task satisfy the high-level policy. In dynamic enforcement, one records the history of who performs which steps in a task instance and determines whether the set of users involved in the task instance satisfies the policy. In the rest of this section, we discuss these two enforcement approaches.

2.2.1 Static Enforcement

Static enforcement can be achieved either directly or indirectly. In direct static enforcement, one verifies whether an access control state is safe with respect to a high-level security policy. In indirect static enforcement, one specifies constraints so that any access control state satisfying the constraints is safe with respect to the policy.

Direct static enforcement of SoD policies, which are a subclass of the policies that can be specified in the algebra, has been studied in [20]. It has been shown that checking whether an access control state statically satisfies an SoD policy, i.e., whether every set of users who together have all the permissions for the task contains at least k users, is $\mathbf{coNP}$-complete [20]. As SoD policies can be specified in the algebra, direct statement enforcement of policies in the algebra requires solving an intractable problem. Computationally expense notwithstanding, we argue that the study of direct enforcement of static high-level policies is necessary for the following reasons. First, direct static enforcement is the most simple and straightforward enforcement mechanism for high-level security policies. Its performance will be used as a benchmark for comparison when evaluating other enforcement mechanisms. Second, even though direct static enforcement is computationally intractable in theory, it is interesting and necessary to study its performance for instances that are likely to occur in practice. Third, direct enforcement cannot be entirely replaced by indirect enforcement. It is oftentimes difficult or even impossible to create efficiently-verifiable constraints to precisely capture a high-level policy. For example, Li et al. studied indirect enforcement by using Static Mutually Exclusive Roles (SMER) to enforce SoD policies in the context of role-based access control (RBAC), and showed that there exist SoD policies such that no set of SMER constraints can *precisely* capture them [20]. Most of the time, the

set of constraints created for a security policy is more restrictive than the policy itself. That is to say, some access control states that are safe with respect to the security policy will be ruled out by the constraints. In situations where precise enforcement is desired, direct enforcement may be the only option.

Direct static enforcement requires solving the Static Safety Checking (SSC) problem, which we formally define through the following definitions.

Definition 2.2.4 (State) An access control system *state* is given by a triple $\langle U, UR, UP \rangle$, where $UR \subseteq U \times \mathcal{R}$ determines user-role memberships and $UP \subseteq U \times \mathcal{P}$ determines user-permission assignment, where $\mathcal{P}$ is the set of all permissions.

Note that a state $\langle U, UR, UP \rangle$ uniquely determines a configuration $\langle U, UR \rangle$ used by term satisfaction. Hence, we may discuss term satisfaction in a state without explicitly mentioning the corresponding configuration. Note also that a user may be assigned a permission directly or indirectly (e.g. via role membership), and the relation UP has taken both ways into consideration.

We say that a userset X *covers* a set P of permissions if and only if the following holds: $\{\, p \mid \exists u \in X[(u, p) \in UP] \,\} \supseteq P$.

Next, we define the notion of safety in direct static enforcement. As we mentioned earlier, the idea of static enforcement is that, by careful design of access control states, one can guarantee that every set of users who together have enough permissions to complete a task satisfies the security policy of the task, and thus runtime checking is unnecessary. While introducing no runtime overhead, static enforcement has a limitation, that is, only monotonic security policies can be enforced statically. The reason is that permission coverage is monotonic with respect to usersets. In other words, if X covers P, then any superset of X also covers P. However, as we emphasized in Section 2.1.5, term satisfaction does not have the monotonicity property. In order to specify monotonic policies, we may use terms in the form of $(\phi \odot \mathsf{All}^+)$. A userset U satisfies $(\phi \odot \mathsf{All}^+)$ if and only if U contains a subset that satisfies ϕ.

When static enforcement is the only enforcement approach, all policies need to be implicitly monotonic to be enforceable. We thus introduce the notion of static safety, which implicitly assumes each term means its monotonic closure.

Definition 2.2.5 (Static Safety) A high-level security policy is given as a pair $\mathsf{sp}\langle P, \phi\rangle$, where $P \subseteq \mathcal{P}$ is a set of permissions and ϕ is a term in the algebra. An access control state $\langle U, UR, UP\rangle$ is *statically safe* with respect to $\mathsf{sp}\langle P, \phi\rangle$, if and only if, for every userset X that covers P, X satisfies the monotonic closure of ϕ (i.e. X satisfies $(\phi \odot \mathsf{All}^{+})$). If a state is statically safe with respect to a policy, we say that it *satisfies* the policy.

Note that in the above definition, we require that, for each userset X that covers P, X satisfies the monotonic closure of ϕ rather than ϕ itself. Equivalently, an access control state is statically safe with respect to $\mathsf{sp}\langle P, \phi\rangle$ if and only if for every userset X that covers P, there exists $X' \subseteq X$, such that X' satisfies ϕ.

The problem of checking static safety is defined as follows; its computational complexity will be studied in Section 2.5.1.

Definition 2.2.6 (SSC) Given a static safety policy $\mathsf{sp}\langle P, \phi\rangle$, the problem of determining whether a given state $\langle U, UR, UP\rangle$ is statically safe with respect to $\mathsf{sp}\langle P, \phi\rangle$ is called the *Static Safety Checking (SSC) problem*.

Note also that Definition 2.2.5 does not require $\langle U, UR, UP\rangle$ to contain a userset that covers P in $\mathsf{sp}\langle P, \phi\rangle$. If a state does not contain any userset that covers P, then it trivially satisfies $\mathsf{sp}\langle P, \phi\rangle$. Checking whether there exists a userset in $\langle U, UR, UP\rangle$ that covers P can be done in linear time with respect to the size of UP.

To check static safety, one needs to determine whether a set of users contains a subset that satisfy a term. This problem is defined as follows.

Definition 2.2.7 (SAFE) Given a term ϕ, a configuration $\langle U, UR\rangle$, and a userset X, a userset X is *safe* with respect to a term ϕ under configuration $\langle U, UR\rangle$, if and only if there exists $X' \subseteq X$ such that X' satisfies ϕ under $\langle U, UR\rangle$.

The *Userset-Term Safety (SAFE) problem* determines whether X is *safe* with respect to a term ϕ under configuration $\langle U, UR\rangle$.

SAFE can be viewed as a special case of UTS, because X is safe with respect to ϕ if and only if X satisfies $(\phi \odot \mathsf{All}^+)$; however, it may be solved more efficiently when treated as a separate problem. The computational complexity of SAFE will be studied in Section 2.5.

We point out that SAFE is technically the same problem as TCSAT, even though they are motivated by different purposes. In SAFE, we ask whether a userset X contains a subset that satisfies ϕ under $\langle U, UR\rangle$, where $X \subseteq U$. Since users in U/X are irrelevant in answering such a question, the problem is equivalent to whether X contains a subset that satisfies ϕ under $\langle X, UR\rangle$, which is the same as whether there is a userset in the configuration $\langle X, UR\rangle$ that satisfies ϕ.

As we mentioned earlier, static enforcement can only enforce security policies with the monotonicity property. To enforce non-monotonic policies, we may use a dynamic enforcement scheme.

2.2.2 Dynamic Enforcement

Similar to static enforcement, dynamic enforcement can be achieved either directly or indirectly as well.

To directly enforce a policy $\langle task, \phi\rangle$, one identifies the steps in performing the task. The system maintains a history of each instance of the task, which includes information on who have performed which steps. For any task instance, one can compute the set of users (denoted as U_{past}) who have performed at least one step of the instance. Before a user u performs a step of the instance, the system checks to ensure that there exists a superset of $U_{past} \cup \{u\}$ that can satisfy ϕ upon finishing all steps of the task. In particular, if u is about to perform the last step of the task instance, it is required by the policy that $U_{past} \cup \{u\}$ satisfies ϕ. As we will see in Section 2.4, checking whether a userset satisfies a term is computationally expensive. In practice, people usually use workflows with security constraints to indirectly enforce high-level security policies.

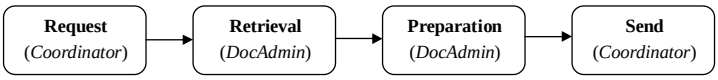
In the rest of this section, we give an example of the secure task design process. The example demonstrates how to use a workflow as an indirect dynamic enforcement scheme

for a high-level security policy specified in the algebra. We would like to point out that in the design of workflows, a designer may take efficiency, quality of service, and other practical restrictions into account in addition to security requirements.

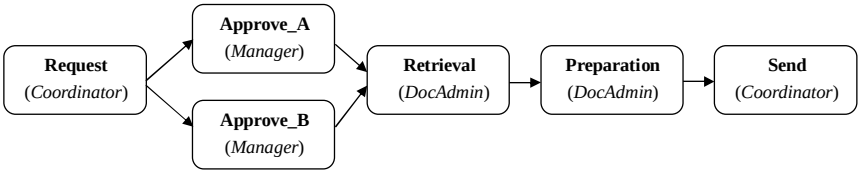
Example 2 Company *XYZ* newly established a plan to share some of its classified documents with its business partners. As the task (denoted as T_s) involves disclosure of classified documents, it is considered to be sensitive by *XYZ* and has to go through a security design procedure. The first step is the high-level policy design, which is performed by a security officer $Alice$. After evaluating the risks and effects of T_s, $Alice$ decides that at least two `Managers` must be involved in the task. She then creates a high-level security policy $(\texttt{Manager} \otimes \texttt{Manager}) \odot \mathsf{All}^+$ for T_s.

The second step is to design a workflow to model T_s in compliance with the high-level security policy. This is performed by a system designer Bob. T_s consists of four physical steps: 1) a business partner coordinator (denoted as `Coordinator`) receives a request from a business partner; 2) a document administrator (denoted as `DocAdmin`) retrieves the document from company archives; 3) a `DocAdmin` performs pre-releasing preparation on the document, such as anonymizing certain items; 4) a `Coordinator` sends the post-preparation document to the business partner. To begin with, Bob creates a workflow W_1 with the four physical steps of T_s, which is shown in Figure 2.1-a. He then introduces two additional steps into W_1 so as to comply with the security policy $(\texttt{Manager} \otimes \texttt{Manager}) \odot \mathsf{All}^+$. He adds two steps to W_1 so that a classified document will not be retrieved until two `Managers` have approved the request on disclosure. Furthermore, in order to provide better quality of service, Bob adds a binding of duty constraint to the workflow so that the coordinator who received the request is responsible to send the document to the business partner. The final workflow W_2 modeling T_s is shown in Figure 2.1-b. It can be verified that any team of users who completes W_2 must satisfy $(\texttt{Manager} \otimes \texttt{Manager}) \odot \mathsf{All}^+$.

It is interesting future work to study how to verify whether a workflow is compliant with a high-level security policy specified in the algebra. In the upcoming sections, we will



a. A workflow (W_1) consisting of physical steps of the task on releasing classified documents to corporate partners. **Request** and **Send** are authorized to the role *Coordinator*, while **Retrieval** and **Preparation** are authorized to *DocAdmin*.



Constraint 1: **Approve_A** and **Approve_B** must be performed by different users

Constraint 2: **Request** and **Send** must be performed by the same user

b. A workflow (W_2) consisting of physical steps, security-oriented steps and constraints in compliance with a high-level security policy. **Approve_A** and **Approve_B** are authorized to *Manager* and may be performed in parallel. Constraint 2 is specified for the purpose of quality of service.

Figure 2.1. Workflows in Example 2

study the computational problems (i.e. TSAT, TCSAT, UTS, SAFE and SSC) defined in this section. As we have seen, these problems are important in the analysis and enforcement of high-level security policies, and are of both theoretical and practical interest.

2.3 Two Term Satisfiability Problems

In this section, we study the computational complexities of TSAT and TCSAT.

2.3.1 The Term Satisfiability (TSAT) Problem

As the algebra supports negation, it is not surprising that unsatisfiable terms exist. A simple example of a term that is not satisfiable is $(r \sqcap \neg r)$. Another source of unsatisfiable terms is the use of explicit sets of users in a term. For example, the term $(\{Alice, Bob\} \sqcap \{Carl\})$ is not satisfiable. However, even if a term does not contain nega-

tion or explicit sets of users, it may still be unsatisfiable. An example of such a term is $\phi = (r_1 \sqcap (r_2 \otimes r_3))$, where r_1, r_2 and r_3 are roles. In the example, r_1 is satisfiable only by a singleton userset, and $(r_2 \otimes r_3)$ is satisfiable only by a userset of cardinality 2. Therefore, there does not exist a userset that satisfies ϕ.

We now show that TSAT is $\mathbf{NP}$-complete in general. We identify the source of intractability by identifying two special cases that are $\mathbf{NP}$-hard. One special case (Lemma 2.3.1 below) involves the negation operator, and the other (Lemma 2.3.2 below) involves explicit sets of users. In Section 2.3.2, we show that for terms that are free of negation and explicit sets of users, TSAT can be efficiently solved.

Lemma 2.3.1 TSAT over terms built using only roles and the operators $\neg$, $\sqcap$, and $\sqcup$ is $\mathbf{NP}$-hard.

Lemma 2.3.2 TSAT over terms built using only explicit sets of users and the operators $\sqcap$, $\sqcup$, and $\odot$ is $\mathbf{NP}$-hard.

To show that TSAT is in $\mathbf{NP}$, we need the following lemma, which shows that if a term is satisfiable, then there exists an evidence of polynomial size.

Lemma 2.3.3 If a term ϕ is satisfiable, then there exists a userset U and a configuration $\langle U, UR \rangle$, such that U satisfies ϕ under $\langle U, UR \rangle$, $|U| \leq |\phi|$ and $|UR| \leq |\phi|^2$, where $|\phi|$ is the number of occurrences of atomic terms in ϕ.

Theorem 2.3.4 TSAT is $\mathbf{NP}$-complete.

Please refer to Appendix A.2 for the proofs of Lemmas 2.3.1, 2.3.2, 2.3.3 and Theorem 2.3.4.

2.3.2 TSAT for the Sub-Algebra Free of Negation and Explicit Sets of Users

Lemmas 2.3.1 and 2.3.2 show that if a term involves negation or explicit sets of users, then determining whether it is satisfiable or not may be intractable. We now study the

term satisfiability problem for terms that are free of explicit sets of users and negation. For convenience, we call such terms *SNF (Set-and-Negation Free) terms*. The following lemma states an important property of terms that are free of negation.

Lemma 2.3.5 Let ϕ be a term that does not contain the operator $\neg$. If userset X satisfies ϕ under configuration $\langle U, UR\rangle$, then X satisfies ϕ under configuration $\langle U, UR'\rangle$, where $UR \subset UR'$.

Lemma 2.3.5 essentially states that, for terms that are free of negation, satisfaction is monotonic with respect to user-role assignment. The proof of the lemma is straightforward and is omitted.

We have the following theorem.

Theorem 2.3.6 Checking whether an SNF term is satisfiable is in $\mathbf{P}$.

To prove Theorem 2.3.6, we first introduce the notion of characteristic sets for SNF terms in Definition 2.3.1. Definition 2.3.1 essentially gives an algorithm to compute the characteristic set of a given SNF term. Then, we show that the algorithm given in Definition 2.3.1 is a polynomial time algorithm. Finally, we prove an important property of characteristic set, that is, an SNF term is satisfiable if and only if its characteristic set is non-empty. To determine whether an SNF term is satisfiable, we can run a polynomial-time algorithm to compute its characteristic set and check whether the characteristic set is empty or not.

To begin with, we introduce the notion of characteristic sets. A key observation is that, in order to satisfy a term, a userset must be of certain size. For example, $(r_1 \odot (r_2 \otimes r_3))$ can be satisfied by a set of 2 or 3 users, but not by a set containing 1 or 4 or any other number of users. We thus call $\{2, 3\}$ the characteristic set of the term $(r_1 \odot (r_2 \otimes r_3))$.

Definition 2.3.1 (Characteristic Set) The *characteristic set* of an SNF term ϕ, which is denoted as $C(\phi)$, is a set of natural numbers computed as follows:

- $C(\mathsf{All}) = C(r) = \{1\}$, where r is a role

- $C(\phi_1 \sqcup \phi_2) = C(\phi_1) \cup C(\phi_2)$
- $C(\phi_1 \sqcap \phi_2) = C(\phi_1) \cap C(\phi_2)$
- $C(\phi^+) = \{i \mid i \in [1, \infty)\}$, where ϕ is a unit term free of explicit sets of users and negations
- $C(\phi_1 \odot \phi_2) = \{i \mid \exists\, c_1 \in C(\phi_1)\ \exists\, c_2 \in C(\phi_2)\, [\, max(c_1, c_2) \leq i \leq c_1 + c_2\,]\}$
- $C(\phi_1 \otimes \phi_2) = \{\ c_1 + c_2 \mid c_1 \in C(\phi_1)\ \wedge\ c_2 \in C(\phi_2)\ \}$

An integer k is called a *characteristic number* of ϕ if and only if $k \in C(\phi)$.

Note that the above definition states how to compute the characteristic set of a given SNF term. As examples, we give the characteristic sets of some terms in below.

- $C(\mathsf{All} \otimes \mathsf{All} \otimes \mathsf{All}) = \{3\}$
- $C(\mathtt{Manager} \odot \mathtt{Accountant}) \otimes \mathtt{Treasurer}) = \{2, 3\}$

  The term $(\mathtt{Manager} \odot \mathtt{Accountant})$ can be satisfied by two users as well as by a single user who is both a `Manager` and an `Accountant`. An additional user is needed to satisfy `Treasurer`.

- $C((\mathtt{Clerk} \sqcup \mathtt{Accountant}) \otimes (\mathtt{Clerk} \sqcap \mathtt{Manager})) = \{2\}$

  One user is required for $(\mathtt{Clerk} \sqcup \mathtt{Accountant})$, and for $(\mathtt{Clerk} \sqcap \mathtt{Manager})$, and the $\otimes$ mandates that the two terms be satisfied by different users.

- $C((\mathtt{Manager} \odot \mathtt{Accountant} \odot \mathtt{Treasurer}) \sqcap \mathtt{Clerk}^+) = \{1, 2, 3\} \cap \{i | i \in [1, \infty)\} = \{1, 2, 3\}$

Given a term ϕ, computing $C(\phi)$ requires at most $2|\phi| - 1$ steps according to the algorithm in Definition 2.3.1, where $|\phi|$ is the number of occurrences of atomic terms in ϕ and ϕ contains $|\phi| - 1$ binary operators. A step in the algorithm may require such operations: set union, set intersection, computing the sums of all pairs of elements from two different sets. If the size of intermediate results (which are sets) is bounded by $|\phi|$, then each step

can be performed in polynomial time, and thus the algorithm finishes in polynomial time. However, when a term contains $^+$, its characteristic set could be an infinite set. Fortunately, the following Lemma 2.3.7 shows that if $C(\phi)$ is an infinite set, it must always contain all the numbers that are greater than $|\phi|$. In this case, we do not have to deal with infinitely many elements in a characteristic set individually, as $\{|\phi|+1, |\phi|+2, \dots\}$ can be treated as one unit during computation.

Lemma 2.3.7 Let ϕ be an SNF term and $|\phi|$ be the number of occurrences of atomic terms in ϕ. One of the following two cases holds:

- $C(\phi) \subseteq \{1, 2, \dots, |\phi|\}$
- $C(\phi) = W \cup \{|\phi|+1, |\phi|+2, \dots\}$, where $W \subseteq \{1, 2, \dots, |\phi|\}$

With Lemma 2.3.7, we can prove the following lemma. The proofs of Lemmas 2.3.7 and 2.3.8 are given in Appendix A.2.2.

Lemma 2.3.8 Given an SNF term ϕ, $C(\phi)$ can be computed in polynomial time with respect to $|\phi|$.

The following theorem states an important property of characteristic sets.

Theorem 2.3.9 Given an SNF term ϕ and a positive integer k, there exists a userset U of size k and a configuration such that U satisfies ϕ under the configuration, if and only if k is a characteristic number of ϕ (i.e. $k \in C(\phi)$).

The proof of Theorem 2.3.9 is given in Appendix A.2.2.

Corollary 2.3.10 An SNF term ϕ is satisfiable if and only if $C(\phi) \neq \emptyset$

With Lemma 2.3.8 and the above corollary, we can see that TSAT over SNF terms is in $\mathbf{P}$.

Another usage of characteristic set is to determine whether a term satisfies some minimal SoD requirements. If the smallest characteristic number of the term is k, then no $k-1$ users can satisfy the term.

Finally, we can extend the notion of characteristic set to non-SNF terms by defining $C(\neg\phi) = \{1\}$, where ϕ is a unit term, and $C(S) = \{1\}$, where S is an explicit set of users. But in that case, it is no longer true that for every integer $k \in C(\phi)$, there is a userset of size k that satisfies ϕ. For example, $C(\{Alice, Bob\} \sqcap \{Carl\}) = C(\{Alice, Bob\}) \cap C(\{Carl\}) = \{1\}$, even though the term $(\{Alice, Bob\} \sqcap \{Carl\})$ is not satisfiable. But it remains true that for any userset X that satisfies a term ϕ, $|X| \in C(\phi)$.

2.3.3 The Term-Configuration Satisfiability (TCSAT) Problem

We have discussed the TSAT problem, which asks whether a term is satisfiable at all. We now examine the TCSAT problem, which asks whether a term is satisfiable under a certain configuration. When a security officer comes up with a term for a high-level security policy of a task, he/she may want to know whether there exists a set of users that satisfies the term and hence is able to perform the task under the current configuration.

Observe that TCSAT is equivalent to TSAT for terms using only explicit sets of users but not roles or the keyword All. Given an instance of TCSAT, which consists of a term ϕ and a configuration $\langle U, UR\rangle$, one can replace each role (or the keyword All) in ϕ with the corresponding set of users in the configuration, which results in a new term ϕ'. In this case, ϕ' is independent of configuration, and ϕ is satisfiable under $\langle U, UR\rangle$ if and only if ϕ' is satisfiable. Therefore, it follows from Lemma 2.3.2 and Theorem 2.3.4 that TCSAT is NP-complete; this is stated in the following theorem.

Theorem 2.3.11 TCSAT is NP-complete.

We mentioned earlier that TCSAT is equivalent to SAFE. In Section 2.5 we will examine the computational complexities of SAFE when only some subsets of operators are allowed. Those results for SAFE apply to TCSAT as well.

Table 2.1
Various sub-cases of the Userset Term Satisfaction (UTS) problem and the corresponding time-complexity

$\neg$	$+$	$\sqcup$	$\sqcap$	$\odot$	$\otimes$	**Complexity**	**Reduction**
✓	✓	✓	✓	✓	✓	**NP**-complete	
		✓		✓		**NP**-complete	Set Covering
		✓			✓	**NP**-complete	Set Packing
			✓	✓		**NP**-complete	Set Covering
			✓		✓	**NP**-complete	Set Covering
				✓	✓	**NP**-complete	Domatic Number
✓	✓	✓	✓			**P**	
✓	✓			✓		**P**	
✓	✓				✓	**P**	

2.4 The Userset-Term Satisfaction (UTS) Problem

In this section, we study the computational complexities of the Userset-Term Satisfaction (UTS) problem, which asks: Given a configuration $\langle U, UR\rangle$, a userset X, and a term ϕ, whether X satisfies ϕ under $\langle U, UR\rangle$? We will show that UTS in the most general case (i.e., arbitrary terms in which all operators are allowed) is **NP**-complete. In order to understand how the operators affect the computational complexities, we consider sub-algebras in which only some subset of the six operators $\{\neg, +, \sqcap, \sqcup, \odot, \otimes\}$ is allowed. For example, $UTS\langle\neg, +, \sqcup, \sqcap\rangle$ denotes the sub-case of UTS where ϕ does not contain operators $\odot$ or $\otimes$, while $UTS\langle\otimes\rangle$ denotes the sub-case of UTS where $\otimes$ is the only kind of operator in ϕ. $UTS\langle\neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ denotes the general case. Observe that unlike in the case of TSAT, whether to allow explicit sets of users in a term or not does not affect the computational complexities of UTS, because a fixed configuration is given in UTS, and one can thus replace each occurrence of a role in the term with the explicit set of the role's members.

Theorem 2.4.1 The computational complexities of UTS and its subcases are given in Table 2.1.

The proof of Theorem 2.4.1 is done in two parts. First, in Appendix A.3.1, we prove that the five cases $UTS\langle \sqcup, \odot \rangle$, $UTS\langle \sqcap, \odot \rangle$, $UTS\langle \sqcup, \otimes \rangle$, $UTS\langle \sqcap, \otimes \rangle$, and $UTS\langle \odot, \otimes \rangle$ are $\mathbf{NP}$-hard by reducing the $\mathbf{NP}$-complete problems SET COVERING, DOMATIC NUMBER, and SET PACKING to them. Second, in Appendix A.3.2, we prove that the general case $UTS\langle \neg, +, \sqcup, \sqcap, \odot, \otimes \rangle$ is in $\mathbf{NP}$. In Section 2.4.1, we identify a wide class of syntactically restricted terms for which the UTS problem is tractable. The class of restricted terms subsumes all the cases listed as in $\mathbf{P}$ in Table 2.1.

2.4.1 UTS is Tractable for Terms in Canonical Forms

From Table 2.1, UTS is $\mathbf{NP}$-complete in all but one sub-algebras that contain at least two binary operators; however, using any one binary operator by itself remains tractable. In this subsection, we show that if a term satisfies certain syntactic restrictions, then even if all operators appear in the term, one can still efficiently determine whether a userset satisfies the term.

Definition 2.4.1 (Canonical Forms for Terms) The canonical forms for terms are defined as follows:

- A term is *in level-1 canonical form* (called a 1CF term) if it is t or t^+, where t is a unit term. Recall that a unit term can use the operators $\neg$, $\sqcap$, and $\sqcup$. We call t the *base* of the 1CF term.
- A term is *in level-2 canonical form* (called a 2CF term) if it consists of one or more sub-terms that are 1CF terms, and these sub-terms are connected only by the operator $\sqcap$.
- A term is *in level-3 canonical form* (called a 3CF term) if it consists of one or more sub-terms that are 2CF terms, and these sub-terms are connected only by the operator $\otimes$.

- A term is *in level-4 canonical form* (called a 4CF term) if it consists of one or more sub-terms that are 3CF terms, and these sub-terms are connected only by the operator $\odot$.

- A term is *in level-5 canonical form* (called a 5CF term) if it consists of one or more sub-terms that are 4CF terms, and these sub-terms are connected only by operators in the set $\{\sqcup, \sqcap\}$.

We say that a term is *in canonical form* if it is in level-5 canonical form. Observe that any term that is in level-i canonical form is also in level-$(i+1)$ canonical form for any $i \in [1, 4]$.

To check whether a term ϕ is in canonical form, one parses ϕ into a syntax tree and then traverses the tree in a depth-first manner to see if any syntactical restriction described in Definition 2.4.1 is violated. This can be done in polynomial time in the size of ϕ.

Theorem 2.4.2 Given a term ϕ in canonical form, a set X of users, and a configuration $\langle U, UR\rangle$, checking whether X satisfies ϕ under $\langle U, UR\rangle$ can be done in polynomial time.

Proof Recall that, by definition, X satisfies $\phi_1 \sqcap \phi_2$ if and only if X satisfies both ϕ_1 and ϕ_2, and X satisfies $\phi_1 \sqcup \phi_2$ if and only if X satisfies either ϕ_1 or ϕ_2. Therefore, to determine whether X satisfies a 5CF term, one can first determine whether X satisfies each of the 4CF sub-terms, and then combine these results using logical conjunction and disjunction.

For a 1CF term ϕ, if it is a unit term, then it is straightforward to determine whether X satisfies ϕ, because a unit term can be satisfied only by a singleton set, and because of the definitions of $\sqcap$ and $\sqcup$. If ϕ is of the form t^+, where t is a unit term, then one just needs to determine whether each user in X satisfies t. Therefore, one can efficiently check whether X satisfies a 1CF term.

Given a 2CF term, if at least one sub-term is a unit term, then one can get an equivalent 1CF term by removing all occurrences of $^+$. For example, $(t_1 \sqcap t_2^+)$ is equivalent to $t_1 \sqcap t_2$. Given a 2CF term where all sub-terms have $^+$, it may be rewritten as an equivalent 1CF term, according to algebraic properties. For example, $(t_1^+ \sqcap t_2^+)$ is equivalent to $(t_1 \sqcap t_2)^+$. Hence, any 2CF term can be transformed into an equivalent 1CF term. We assume that the

transformation is performed whenever applicable so that we don't need to consider 2CF terms explicitly.

Given a 3CF term $P = (\phi_1 \otimes \cdots \otimes \phi_m)$, where each ϕ_i is a 1CF term. Let us first consider a special case that each ϕ_i is a unit term t_i. In this case, one can determine whether X satisfies ϕ_i by solving the following bipartite graph maximal matching problem. One constructs a bipartite graph such that one set of nodes consists of users in X and the other consists of the m unit terms $t_1, t_2, \ldots, t_m$; and there is an edge between $u \in X$ and t_i if and only if $\{u\}$ satisfies t_i. One then computes a maximal matching of the graph (which can be done in polynomial time); if the size of the matching is $\max(|X|, m)$, then X satisfies P; otherwise, X does not satisfy P.

The case that a 3CF term contains $+$ is more complicated, as is the case for a 4CF term. The proof for the 4CF case (which subsumes the 3CF case) is long and offers limited new insights. We thus leave the proof in Appendix A.3.3. ■

Terms in canonical forms appear to be general enough to specify many high-level security policies in practice. We arrive at these canonical forms by excluding the intractable cases used in the **NP**-hardness proofs, and by studying how to efficiently handle terms involving the binary operators.

2.5 The Userset-Term Safety (SAFE) Problem and the Static Safety Checking (SSC) Problem

In this section, we study the Userset-Term Safety (SAFE) problem and the Static Safety Checking (SSC) problem. As we have pointed out in Section 2.3.3, SAFE is technically equivalent to TCSAT, even though the two problems are motivated by different purposes. Since TCSAT is **NP**-complete, SAFE is **NP**-complete in general.

Also, SAFE is related to yet different from UTS. SAFE asks whether X is safe with respect to a term ϕ under a configuration; this is monotonic in that if X is safe, then any superset of X is also safe. However, UTS is not monotonic. This difference has subtle but important effects. For example, under SAFE, the operator $\odot$ is equivalent to logical

conjunction, that is, X is safe with respect to $\phi_1 \odot \phi_2$ if and only if X is safe with respect to both ϕ_1 and ϕ_2. This is because X is safe with respect to $\phi_1 \odot \phi_2$ if and only if X contains a subset X_0 that is the union of two subsets X_1 and X_2 such that X_1 satisfies ϕ_1 and X_2 satisfies ϕ_2. This is equivalent to X containing two subsets X_1 and X_2 such that X_1 satisfies ϕ_1 and X_2 satisfies ϕ_2. In contrast, the operator $\odot$ is different from logical conjunction under UTS. That X satisfies $\phi_1 \odot \phi_2$ does not imply X satisfies both ϕ_1 and ϕ_2. For example, $\{u_1, u_2\}$ satisfies $\mathsf{All} \odot \mathsf{All}$, but does not satisfy All, because term satisfaction is not monotonic. Another difference regards the operator $\sqcap$. The operator $\sqcap$ is equivalent to logical conjunction under UTS, by definition of term satisfaction. However, $\sqcap$ is stronger than logical conjunction under SAFE. That X is safe with respect to $\phi_1 \sqcap \phi_2$ implies that X is safe with respect to both ϕ_1 and ϕ_2, but the other direction is not true. For example, given $UR = \{(u_1, r_1), (u_2, r_2)\}$, $X = \{u_1, u_2\}$ is safe with respect to both r_1 and r_2, but is not safe with respect to $r_1 \sqcap r_2$.

Because of these and other differences, the computational complexity results about UTS do not imply computational complexity results for SAFE. In the rest of this section, we give the computational complexities of SAFE and its subcases, and compare them with those of UTS. Similar to the discussion of UTS in Section 2.4, we consider all sub-algebras in which only some subset of the six operators in $\{\neg, +, \sqcap, \sqcup, \odot, \otimes\}$ is allowed.

Theorem 2.5.1 The computational complexities of SAFE and its subcases are given in Table 2.2.

Please refer to Appendix A.4 for proofs of the above theorem. In the appendix, we first prove that the three cases $SAFE\langle\neg, +, \sqcap, \sqcup\rangle$, $SAFE\langle\neg, +, \sqcup, \odot\rangle$, and $SAFE\langle\neg, +, \otimes\rangle$ are in $\mathbf{P}$. As we mentioned at the beginning of the section, SAFE is $\mathbf{NP}$-complete in general, which implies that all of its subcases are in $\mathbf{NP}$. Hence, to prove all the $\mathbf{NP}$-completeness results, it suffices to prove that the four cases $SAFE\langle\sqcap, \odot\rangle$, $SAFE\langle\sqcup, \otimes\rangle$, $SAFE\langle\sqcap, \otimes\rangle$, and $SAFE\langle\odot, \otimes\rangle$ are $\mathbf{NP}$-hard.

Comparing Table 2.2 with Table 2.1, we found that the computational complexities of all subcases of SAFE are the same as those of UTS except for the subcase in which only

Table 2.2
Various sub-cases of the Userset-Term Safety (SAFE) problem and the corresponding time-complexity

$\neg$	$+$	$\sqcup$	$\sqcap$	$\odot$	$\otimes$	**Complexity**	**Reduction**
✓	✓	✓	✓	✓	✓	**NP**-complete	
		✓			✓	**NP**-complete	Set Packing
			✓	✓		**NP**-complete	Set Covering
			✓		✓	**NP**-complete	Set Covering
				✓	✓	**NP**-complete	Domatic Number
✓	✓	✓	✓			**P**	
✓	✓	✓		✓		**P**	
✓	✓				✓	**P**	

operators in $\{\neg, +, \sqcup, \odot\}$ are allowed. $SAFE\langle\neg, +, \sqcup, \odot\rangle$ is in **P**, while $UTS\langle\sqcup, \odot\rangle$ is **NP**-hard. Intuitively, $UTS\langle\sqcup, \odot\rangle$ is computationally more expensive than SAFE $\{\sqcup, \odot\}$ for the following reason: given a term $\phi = (\phi_1 \odot \cdots \odot \phi_m)$ and a userset U, U is safe with respect to ϕ if and only if U is safe with respect to ϕ_i for every $i \in [1, m]$. In other words, for SAFE, one may check whether U is safe with respect to ϕ_i independently from ϕ_j $(i \neq j)$. However, when it comes to UTS, such independency no longer exists and one has to take into account whether every user in U is used to satisfy some ϕ_i in the term ϕ.

2.5.1 Static Safety Checking (SSC) Problem

Given a high-level security policy $\mathsf{sp}\langle P, \phi\rangle$, the Static Safety Checking (SSC) problem asks whether a given state $\langle U, UR, UP\rangle$ is statically safe with respect to $\mathsf{sp}\langle P, \phi\rangle$. We study the computational complexities of SSC, and consider all subcases where only some subset of the operators in $\{\neg, +, \sqcap, \sqcup, \odot, \otimes\}$ is allowed. We show that the general case of SSC is both **NP**-hard and **coNP**-hard and is in $\mathbf{coNP^{NP}}$, which is a complexity class in Polynomial Hierarchy. The proof of the following theorem is given in Appendix A.5.

Table 2.3
Various sub-cases of the Static Safety Checking (SSC) problem and the corresponding time-complexity

$\neg$	$+$	$\sqcup$	$\sqcap$	$\odot$	$\otimes$	**Complexity**	**Reduction**
✓	✓	✓	✓	✓	✓	**NP**-hard, **coNP**-hard, in $\mathbf{coNP^{NP}}$	
		✓		✓		**coNP**-hard	Validity
			✓	✓		**NP**-hard	$SAFE\langle\sqcap, \odot\rangle$
					✓	**coNP**-complete	Set Covering
✓	✓	✓	✓			**P**	
✓	✓			✓		**P**	

Theorem 2.5.2 The computational complexities of SSC and its subcases are given in Table 2.3.

2.6 Discussions

In this section we discuss potential extensions to the syntax of the algebra, the relationship between the algebra and regular expressions, and the limitations of the algebra's expressive power.

2.6.1 Extensions to the Syntax of the Algebra

In this chapter, we have defined the basic operators in the algebra and examined their properties. We now discuss some additional operators that could be added to the algebra as syntactic sugars.

As discussed in Section 2.1.5, SoD policies are monotonic, as are policies in McLean's formulation of N-person policies [19]; our algebra supports both monotonic policies and policies that are not monotonic. To express a monotonic policy that requires a task to be

performed by a userset that either satisfies a term ϕ or contains a subset that satisfies ϕ, one can use $(\phi \odot \mathsf{All}^+)$. As monotonic policies may be quite common, we introduce a unary operator $\bigtriangledown$ as a syntactic sugar. That is, $\bigtriangledown\phi$ is defined to be $(\phi \odot \mathsf{All}^+)$.

Besides monotonic policies, another type of policy mentioned in Section 2.1.5 states that every user involved in a task must satisfy certain requirements and there need to be at least a certain number of users involved. Let ϕ be a unit term that expresses the requirements. A policy that requires two or more users that satisfy ϕ can be expressed as $((\phi \otimes \phi) \odot \phi^+)$. To simplify the expression of these policies, we define ϕ^{2+} as a syntactic sugar for $((\phi \otimes \phi) \odot \phi^+)$. In general, ϕ^{k+} means that at least k $(k \geq 2)$ users are required and every user involved must satisfy ϕ.

Similar to the above, ϕ^k is a syntactic sugar for a term using operator $\otimes$ to connect k unit terms ϕ. For instance, $\texttt{Accountant}^3$ is defined as $(\texttt{Accountant} \otimes \texttt{Accountant} \otimes \texttt{Accountant})$. More generally, ϕ^k states that exactly k users are required and every user involved must satisfy ϕ. Writing a term in ϕ^k rather than $(\phi \otimes \cdots \otimes \phi)$ explicitly states that all the k sub-terms connected together by $\otimes$ are the same. This makes the policy more succinct and easier to process.

2.6.2 Relationship with Regular Expressions

The syntax of terms in our algebra may remind readers of regular expressions. A regular expression is a string that describes or matches a set of strings, while a term in the algebra is a string that describes or matches a set of sets. Given an alphabet, a regular expression evaluates to *a set of strings*. Given a configuration, a term in our algebra evaluates to *a set of sets*. In the following, we compare our algebra with regular expressions.

For example, the regular expression "$a(b|c)[\hat{}abc]^+$" matches all strings that start with the letter a, followed by either b or c, and then by one or more symbols that are not in $\{a, b, c\}$. A term that is close in spirit to the regular expression is $\{a\} \otimes (\{b\} \sqcup \{c\}) \otimes (\neg\{a, b, c\})^+$, which is satisfied by all sets that contain a, either b or c, and one or more symbols that are not in $\{a, b, c\}$.

From the example, one can draw some analogies between the operators in regular expressions and the ones in our algebra. The operator $|$ in regular expressions is similar to $\sqcup$. Concatenation in regular expression may seem to be related to $\otimes$. One clear difference is that concatenation is order sensitive, whereas $\otimes$ is not, because a string is order sensitive but a set is not. A more subtle difference comes from the property that $\otimes$ requires the two sub-terms be satisfied by disjoint sets. For instance, $\{a\} \otimes \{a\}$ cannot be satisfied by any set. The usage of negation in regular expressions is similar to negation in the algebra; in both cases, negation can be applied only to an expression corresponding to a single element. In regular expression, the closure operator ($*$ or $+$) can be applied to arbitrary sub-expressions. Our algebra requires that repetition (using operator $^+$) can only be applied to unit terms. As we discussed in Section 2.1.5, since the algebra is proposed for security policy specification, we impose such restriction so as to clearly capture real-word security requirements. If the algebra is used in areas other than security policy specification, it is certainly possible to release such restriction so that the algebra can define a wider range of sets. The remaining binary operators $\odot$ and $\sqcap$ have the flavor of set intersection, which does not have counterparts in regular expressions.

Observe that determining whether a string satisfies a regular expression is in $\mathbf{NL}$-complete, where $\mathbf{NL}$ stands for Nondeterministic Logarithmic-Space, and is contained in $\mathbf{P}$. On the other hand, determining whether a userset satisfies a term is $\mathbf{NP}$-complete, even if the term uses only $\sqcup$ and $\otimes$ or only $\sqcup$ and $\odot$. It appears that this increase in complexity is due to the unordered nature of sets. Checking a string against a regular expression can be performed from the beginning of a string to its end; on the other hand, there is no such order in checking a set against a term in the algebra.

As a fundamental tool for defining sets of strings, regular expression is used in many areas. Analogically, because our algebra is about the fundamental concept of defining sets of sets, we conjecture that, besides expression of security policies, the algebra could be used in other areas where set specification is desired. For example, we may use the algebra to specify some sorts of reaction formula, in which each element must have certain properties and in some cases we may be able to choose among several properties. For

another example, our algebra could be used to specify digital-right-management licenses that entitle users to play a set of songs. An example of such licenses is, Alice can play a song in Album A once, and two other songs in either Album B or Album C. Barth and Mitchell studied how to specify such licenses using linear logic in [21].

2.6.3 Limitations of the Algebra's Expressive Power

It is well-known that using regular expression, one cannot express languages that require counting to an unbounded number; for example, one cannot express all strings over the alphabet $\{a, b\}$ that contain the same number of a's as of b's.

Similarly, the algebra as defined in Section 2.1.1 cannot express a policy that requires a set of users in which the number of members of r_1 equals the number of members of r_2. The proof is similar to that of the Pumping Lemma in regular language. We illustrate the sketch of the proof here. Assume, for the purpose of contradiction, that there exists a term ϕ in the algebra that is satisfied only by usersets with an equal number of members of r_1 and members of r_2. Let X_1 be a userset consisting of n users who are members of r_1 but not r_2, and X_2 be another userset consisting of n users who are members of r_2 but not r_1, where $n > |\phi|$. Let $X = X_1 \cup X_2$. By assumption, X satisfies ϕ. Let T be the satisfaction tree of ϕ, whose root is labeled with X. Since $|X| > |\phi|$, T must have leaves corresponding to sub-terms in the form of ϕ_0^+. Also, since $n > |\phi|$, there must exist a leaf N_1 with $\phi_1 = \phi_2^+$ and $L_T(N_1)$ contains a user $u \in X_1$ that does not appear in usersets labeling leaves without $^+$. We may now "pump" (i.e. add) another m copies of u to every node in T whose associated userset contains u. By following the rules in Definition 2.1.4, it can be proved that the tree T', which is acquired from T after pumping, is a satisfaction tree of ϕ. Note that the root of T' is labeled with X', which contains $n + m$ members of r_1 and n members of r_2. According to Theorem 2.1.2, X' satisfies ϕ, which is a contradiction to the assumption.

If we allow the application of $^+$ to non-unit terms and define it as follows:

$$\phi^+ \stackrel{def}{=} \phi \sqcup (\phi \otimes \phi) \sqcup (\phi \otimes \phi \otimes \phi) \sqcup \dots$$

then we can express the policy that requires an equal number of members of r_1 and members of r_2 using the term

$$[(r_1 \sqcap r_2) \sqcup ((r_1 \sqcap \neg r_2) \otimes (r_2 \sqcap \neg r_1)) \sqcup (\neg r_1 \sqcap \neg r_2)]^+$$

Note that the subterm $((r_1 \sqcap \neg r_2) \otimes (r_2 \sqcap \neg r_1))$ matches one user who is a member of r_1 but not r_2 with a user who is a member of r_2 but not r_1.

Even with the extension, however, there are sets of usersets that cannot be expressed. For example, one cannot express a policy that requires that the number of users who are r_1 equals the square of the number of users who are r_2. [2] Further discussions of expressive power and more general algebras are interesting future research topics.

[2]Intuitively, since $^+$ does not record the number of users, there is no way for a term to compute the square of the number of users in a userset.

3 RESILIENCY POLICIES IN ACCESS CONTROL

In the last chapter, we have introduced an algebra for high-level security policy specification. Similar to most existing work on access control policy specification and analysis, our work on the algebra focuses on security properties which ensure that users who should not have access do not get access. However, an equally important aspect of access control is to enable access (selectively).

In this chapter, we introduce the notion of resiliency policies which state properties about enabling access in access control. Resiliency policies require that the access control state is resilient to absent users. For example, the access control system of an institution has three separate permissions regarding release of funds: one permission is an endorsement that the request for funds is legitimate, the second permission is the issuance of a check, and the third one is for logging the transaction. The institution's financial office, which takes charge of funding, is composed of a senior treasurer and a number of junior treasurers. In compliance of the separation of duty principle, the senior treasurer has all permissions except the one for logging, while each of the junior treasurers has only one of the three permissions. As issuing funds is a critical task, the institution would like to ensure that even if a few (e.g., two) treasurers (that may include the senior treasurer) are absent (e.g., due to sickness), the remaining personnel in the financial office still have enough privileges to release funds.

Another example resiliency policy requirement is as follows: There must exist three mutually disjoint sets of users such that each set has no more than four users and the users in each set together have all permissions to carry out a critical task. Such a policy would be needed when one needs to be able to send up to three teams of users to different sites to perform a certain task, perhaps in response to some events. One needs to ensure that each team has enough permissions to perform the task, and each team consists of no more than four users (e.g., due to the limit of transportation means).

Such policies are particularly useful when evaluating whether the access control configuration of a system is ready for emergency response. These policies ensure that even when emergency situations cause some users to be absent, there still exist independent teams of users that have the necessary permissions for carrying out critical tasks. In other words, these policies mandate that there is a certain level of redundancy in assigning permissions to users so that the system can tolerate some users being absent.

The remainder of this chapter is organized as follows. In Section 3.1, we define resiliency policies and the Resiliency Checking problem. We present computational complexities of the Resiliency Checking problem in Section 3.2. Finally, we explore the policy consistency problem, in Section 3.3

3.1 Resiliency Policies and the Resiliency Checking Problem

Definition 3.1.1 (Resiliency Policies) A resiliency policy takes the form

$$\mathsf{rp}\langle P, s, d, t\rangle$$

where rp is a keyword, $P = \{p_1, \ldots, p_n\}$ is a set of permissions, $s \geq 0$ and $d \geq 1$ are integers, and t is either a positive integer or the special symbol ∞.

We say that an access control state satisfies such a resiliency policy if and only if upon removal of any set of s users, there still exist d mutually disjoint sets of users such that each set contains no more than t users and the users in each set together are authorized for all permissions in P.

Example 3 Consider the access control state from Figure 3.1. To issue funds, all three permissions $Endorse,\ Issue$ and Log must be possessed by a set of users. In our resiliency policy, we set $P = \{Endorse,\ Issue,\ Log\}$. If we set $s = 1$ in our policy, then we want the system to be resilient to the absence of any (one) user. If we set $d = 2$, this means that we require two sets of users such that users in each set together possess all permissions. If we set $t = \infty$, this means that the set of users that together possess all permissions can be of any size.

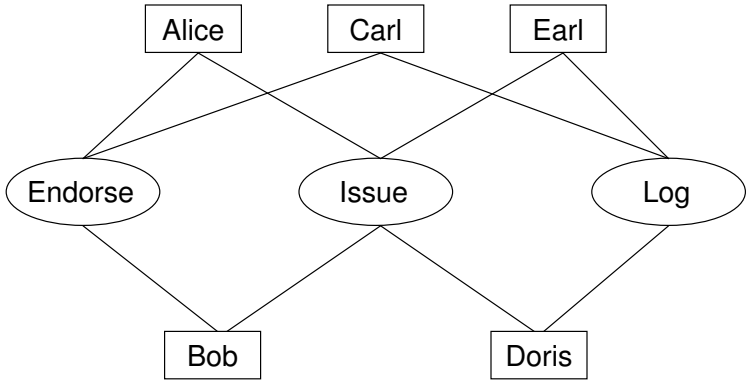


Figure 3.1. An example of an access control state with five users and three permissions

We observe that in our example, $\mathsf{rp}\langle P, 1, 2, \infty\rangle$ is satisfied. For instance, after removing $Alice$, the two users $Carl$ and $Earl$ together have all three permissions, as are Bob and $Doris$. The cases in which another user is removed can be verified similarly. However, $\mathsf{rp}\langle P, 2, 2, \infty\rangle$ is not satisfied because if $Alice$ and Bob are absent, the only user that possesses $Endorse$ is $Carl$, and one user cannot belong to two disjoint sets. Similarly, $\mathsf{rp}\langle P, 2, 1, \infty\rangle$ is satisfied, but $\mathsf{rp}\langle P, 3, 1, \infty\rangle$ is not satisfied because if $Alice$, Bob and $Carl$ are absent, then no user possesses $Endorse$. And finally, we observe that $\mathsf{rp}\langle P, 1, 1, 2\rangle$ is satisfied, but not $\mathsf{rp}\langle P, 1, 1, 1\rangle$ because for the latter case, there exists no single user that has all three permissions.

Intuitively, a resiliency policy $\mathsf{rp}\langle P, s, d, t\rangle$ specifies a fault tolerance requirement with respect to a certain critical task. The set P includes all permissions that are needed to carry out the task. The faults that we would like to tolerate are absent users. The parameter s specifies the number of absent users that we want to be able to tolerate. The parameter d is motivated by the requirement that several teams may be needed to carry out multiple instances of the task. If only one team is needed, then d can be set to 1. The parameter t specifies the size limit of each team. This is motivated by limitations on the maximal number of users that can be involved in any instance of task. If no such limitation exists, then t can be set to ∞.

The two parameters s and d are related. If an access control state satisfies $\mathsf{rp}\langle P, s, d, t\rangle$, then it also satisfies $\mathsf{rp}\langle P, s+i, d-i, t\rangle$ for any i such that $0 < i < d$. For example, if, after removing any 2 users, there exist 3 mutually disjoint sets of users such that each set covers all permissions in P, then after removing any 3 users, there are at least 2 sets left. However, if a state satisfies $\mathsf{rp}\langle P, s+1, d-1, t\rangle$, it may not satisfy $\mathsf{rp}\langle P, s, d, t\rangle$. For our example shown in Figure 3.1, we observe that $\mathsf{rp}\langle P, 1, 2, \infty\rangle$ is satisfied. However $\mathsf{rp}\langle P, 0, 3, \infty\rangle$ is not satisfied because we need the 3 users $Alice$, Bob and $Carl$ that possess $Endorse$ to belong to distinct sets; this still leaves one permission that needs to covered by each set, and we have only two users that remain.

Resiliency policies can be defined in any access control system in which there are users and permissions. This includes almost all access control systems, including Discretionary Access Control systems [22, 23] and Role Based Access Control systems [18]. We assume that an access control state is given by a binary relation $UP \subseteq \mathcal{U} \times \mathcal{P}$, where $\mathcal{U}$ represents the set of all users, and $\mathcal{P}$ represents the set of all permissions. Note that by assuming that a state is given by a binary relation $UP \subseteq \mathcal{U} \times \mathcal{P}$, we are not assuming permissions are directly assigned to users; rather, we assume only that one can calculate the relation UP from the access control state.

Definition 3.1.2 (Resiliency Checking Problem (RCP)) Given a resiliency policy r and an access control state UP, determining whether UP satisfies r is called the Resiliency Checking Problem (RCP).

A resiliency policy has three parameters: s, d, and t. In some situations, one may need to consider only those policies with one or more of these parameters degenerated. The parameter s, which denotes the number of absent users that the system needs to tolerate, may be degenerated to always be 0. The parameter d, which denotes the number of sets of users required, may be degenerated to always be 1. Finally, the parameter t, which denotes the size bound on each set, may be degenerated to always be ∞. There are eight cases where some of the three parameters are degenerated. For example, a resiliency policy in the subcase $RCP\langle s = 0, d = 1\rangle$ has the form $\mathsf{rp}(P, 0, 1, t)$, which asks whether there exists

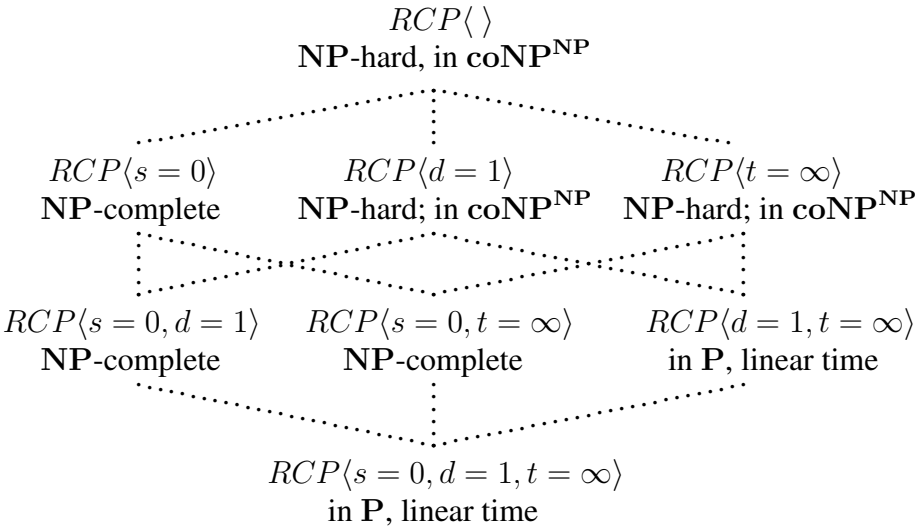


Figure 3.2. Time complexity of the Resiliency Checking Problem (RCP) and its various subcases.

a set of users of size at most t that together have all permissions in P; while the subcase $RCP\langle t=\infty\rangle$ asks whether there exist several distinct sets of users (d sets) each of whose users together have all permissions in P, even after any set of s users is removed from the state. In particular, $RCP\langle\ \rangle$ is the general case of the problem.

3.2 Computational Complexities of the Resiliency Checking Problem

The following theorem summarizes the computational complexity results for RCP and its various subcases. These results are also shown in Figure 3.2.

Theorem 3.2.1 The computational complexities of the Resiliency Policy Checking problem are as follows.

- $RCP\langle\ \rangle$, the most general case, is **NP**-hard and is in $\mathbf{coNP^{NP}}$, as are the two subcases $RCP\langle d=1\rangle$ and $RCP\langle t=\infty\rangle$.
- $RCP\langle s=0, d=1\rangle$, $RCP\langle s=0, t=\infty\rangle$, and $RCP\langle s=0\rangle$ are **NP**-complete.
- $RCP\langle d=1, t=\infty\rangle$ and $RCP\langle s=0, d=1, t=\infty\rangle$ can be solved in linear time.

Our complexity results show that RCP is in $\mathbf{coNP^{NP}}$. This means that the complement of RCP can be solved by a nondeterministic Oracle Turing Machine that has oracle access to a machine that can answer any $\mathbf{NP}$ queries. Intuitively, given an access control state and a resiliency policy $r = \mathsf{rp}(P, s, d, t)$, to decide nondeterministically that the state does not satisfy r, one can guess a set of s users to be removed, and then query the $\mathbf{NP}$ oracle whether the remaining users contain d mutually disjoint sets of users such that each set is of size at most t and the users in each set together have all the permissions in P.

Another way to understand the computational complexity of RCP is to observe that an RCP instance has the form $\forall$ size-s subset, $\exists d$ sets of users that satisfy some requirements that can be efficiently verified. Problems in $\mathbf{NP}$ have the form of $\exists$ an evidence that satisfies some polynomial-time verifiable requirements. Problems in $\mathbf{coNP}$ has the form $\forall$ choices, some polynomial-time verifiable requirements hold. RCP has one alternation of $\forall$ followed by $\exists$, which makes it in $\mathbf{coNP^{NP}}$.

We have shown that RCP (and its two subcases $RCP\langle d = 1\rangle$ and $RCP\langle t = \infty\rangle$) are $\mathbf{NP}$-hard and are in $\mathbf{coNP^{NP}}$. It remains open whether these three problems are $\mathbf{coNP^{NP}}$-complete or not. Readers who are familiar with computational complexity theory will recognize that $\mathbf{coNP^{NP}}$ is a complexity class in the Polynomial Hierarchy. Because the Polynomial Hierarchy collapses when $\mathbf{P} = \mathbf{NP}$, showing that an $\mathbf{NP}$-hard decision problem is in the Polynomial Hierarchy, although is not equivalent to showing that the problem is $\mathbf{NP}$-complete, has the same consequence: the problem can be solved in polynomial time if and only if $\mathbf{P}$ =$\mathbf{NP}$.

In the rest of this section, we prove the results in Theorem 3.2.1. The following lemmas prove that $RCP\langle s = 0\rangle$ is in $\mathbf{NP}$, $RCP\langle s = 0, d = 1\rangle$ and $RCP\langle s = 0, t = \infty\rangle$ are $\mathbf{NP}$-hard, $RCP\langle\ \rangle$ is in $\mathbf{coNP^{NP}}$, and $RCP\langle d = 1, t = \infty\rangle$ is in $\mathbf{P}$. The complexities of other subcases can be implied from these results.

Lemma 3.2.2 $RCP\langle s = 0\rangle$ is in $\mathbf{NP}$.

Proof An instance consists of an access control state UP and a policy $\mathsf{rp}\langle P, 0, d, t\rangle$. UP satisfies $\mathsf{rp}\langle P, 0, d, t\rangle$ if and only if there exist d mutually disjoint sets of users such that the

users in each set together cover all permissions in P and each set has at most t users. If these d sets are given, they can be verified in polynomial time. Therefore, $RCP\langle s = 0\rangle$ is in **NP**. ■

Lemma 3.2.3 $RCP\langle s = 0, d = 1\rangle$ is **NP**-hard.

Proof We reduce the **NP**-complete SET COVERING problem [24] (also referred to as MINIMUM COVERING problem in [25]) to $RCP\langle s = 0, d = 1\rangle$. In SET COVERING, we are given a set S, n subsets of S: $S_1, \ldots, S_n$, and a budget K, and need to determine whether the union of K subsets is the same as S. An instance of $RCP\langle s = 0, d = 1\rangle$ asks whether an access control state UP satisfies a policy $\mathsf{rp}\langle P, 0, 1, t\rangle$. In our reduction, each element in S is mapped to a permission in P and each subset S_i is mapped to a user u_i. In other words, if the subset S_i contains an element, then u_i is authorized for the permission corresponding to the element. We now argue that the mapping ensures that there exists a set of users of size at most K together have all the permissions in P if and only if K subsets cover S. Assume that a set of users of size at most K exists such that those users together have all the permissions in P. Then, we pick the subsets that are mapped to those users, and their union gives us S. For the other direction, assume that K subsets cover S. Then, the K users to which the subsets are mapped together have all the permissions in P. ■

Lemma 3.2.4 $RCP\langle s = 0, t = \infty\rangle$ is **NP**-hard.

Proof We reduce the **NP**-complete DOMATIC NUMBER problem [25] to $RCP\langle s = 0, t = \infty\rangle$. Given a graph $G(V, E)$, the DOMATIC NUMBER problem asks whether V can be partitioned into k disjoint sets $V_1, V_2, \cdots, V_k$, such that each V_i is a dominating set for G. V' is a dominating set for $G = (V, E)$ if for every node u in $V - V'$, there is a node v in V' such that $(u, v) \in E$. An instance of $RCP\langle s = 0, t = \infty\rangle$ asks whether an access control state UP satisfies a policy $\mathsf{rp}\langle P, 0, d, \infty\rangle$. Given a graph $G = (V, E)$, we construct an access control state UP with n users $u_1, u_2, \cdots, u_n$ and n permissions $p_1, p_2, \cdots, p_n$, where n is the number of nodes in V. Each user corresponds to a node in G, and $v(u_i)$ denotes the node corresponding to user u_i. In UP, user u_i is authorized for the permission

p_j if and only if either $i = j$ or $(v(u_i), v(u_j)) \in E$. Let P denote the set $\{p_1, p_2, \cdots, p_n\}$. A dominating set in G corresponds to a set of users that together have all the permissions in P. UP satisfies $\mathsf{rp}\langle P, 0, k, \infty\rangle$ if and only if V contains k disjoint dominating sets. ■

Lemma 3.2.5 $RCP\langle\,\rangle$ is in $\mathbf{coNP^{NP}}$.

Proof We show that the complement of $RCP\langle\,\rangle$ is in $\mathbf{NP^{NP}}$. Assume that we have an oracle that decides the Resiliency Checking problem when $s = 0$, which, as we know, is $\mathbf{NP}$-complete. We construct a nondeterministic oracle Turing machine M that accepts UP and $\mathsf{rp}\langle P, s, d, t\rangle$ when UP does not satisfy $\mathsf{rp}\langle P, s, d, t\rangle$. M nondeterministically removes s users, and then queries the oracle. If the oracle machine returns "yes", M rejects; otherwise, M accepts, because it has found a set of users, the removal of which violates the Resiliency policy. The construction of M shows that the complement of $RCP\langle\,\rangle$ is in $\mathbf{NP^{NP}}$. Therefore, $RCP\langle\,\rangle$ is in $\mathbf{coNP^{NP}}$. ■

Lemma 3.2.6 $RCP\langle d = 1, t = \infty\rangle$ can be solved in linear time.

Proof An instance in $RCP\langle d = 1, t = \infty\rangle$ asks whether an access control state satisfies a policy $\mathsf{rp}\langle P, s, 1, \infty\rangle$. We observe that the answer is "no" if and only if some permission in P is assigned to no more than s users. In this case, removing the s users who have that permission would result in no user having that permission. On the other hand, if each permission is assigned to at least $s + 1$ users, after removing any set of s users, each permission is still assigned to at least one user, which means that the set of all remaining users together have all the permissions in P. ■

Definition 3.2.1 (The Tolerance Bound) Given an access control state UP and a set $\{p_1, \cdots, p_m\}$ of permissions, we define the *tolerance bound* of UP and $\{p_1, \cdots, p_m\}$, denoted by $tb(UP, \{p_1, \cdots, p_m\})$, to be $\min_{1 \le i \le m} \#(p_i)$, where $\#(p_i)$ denotes the number of users who are authorized for p_i in the state UP.

Given an $RCP\langle d = 1, t = \infty\rangle$ instance that asks whether UP satisfies $\mathsf{rp}\langle P, s, 1, \infty\rangle$, the answer is yes if and only if the tolerance bound is at least $s + 1$. More generally, given

an RCP instance that asks whether UP satisfies $\mathsf{rp}\langle P, s, d, t\rangle$, if $s + d > tb(UP, P)$, then the answer is "no". On the other hand, when $d \geq 2$ and $s + d \leq tb(UP, P)$, we do not immediately know whether UP satisfies $\mathsf{rp}\langle P, s, d, t\rangle$ or not.

We now give a linear-time algorithm for calculating the tolerance bound. This, together with the above observations, suffices to prove Lemma 3.2.6. The algorithm maintains a counter for each permission. It first goes through all pairs in UP to count how many users each permission is assigned to. It then returns the minimal value among the counters.

3.3 The Consistency of Resiliency and Separation of Duty Policies

As we have discussed earlier, resiliency policies are a natural complement to traditional safety policies in access control. Consequently, a question arises regarding the consistency of resiliency policies with other policies. In this section, we explore the consistency of resiliency policies and static separation of duty (SSoD) policies.

The intent of an SSoD policy is to preclude any group of users from possessing too many permissions. We adopt the concrete formulation of such policies from Li et al. [26]. An SSoD policy is of the form $\mathsf{ssod}\langle P, k\rangle$, where P is a set of permissions and $1 < k \leq |P|$ is an integer. An access control state satisfies the policy if there exists no set of fewer than k users that together possess all permissions in P. In the policy $\mathsf{ssod}\langle P, k\rangle$, P denotes the set of permissions that are needed to perform a sensitive task, and k denotes the minimal number of users that are allowed to perform the task. If the policy is satisfied, then no set of $k-1$ users can together perform the task, because they do not have all the permissions; thus at least k users need to be involved, achieving the goal of separation of duty. For example, the policy $\mathsf{ssod}\langle\{p_1, p_2\}, 2\rangle$ means that no single user is allowed to have both p_1 and p_2.

In many cases, it is desirable for an access control system to have both resiliency and SSoD policies. If an access control system has only resiliency policies, then they can be satisfied by giving all permissions to all users, resulting in each single user can perform any task. Similarly, if an access control system has only SSoD policies, then they can be satisfied by not giving any permission to any user, resulting in no task can be performed.

It is clear that neither kind of policies by itself is sufficient to capture the security requirements. When both kinds of policies coexist, safety and functionality requirements can all be specified.

Due to their opposite focus, resiliency policies and separation of duty policies can conflict with each other. For example, a separation of duty policy $\mathsf{ssod}\langle P, 2\rangle$ requires that no user possess all permissions in P. A resiliency policy $\mathsf{rp}\langle P, s, d, 1\rangle$ requires the existence of a user that has all permissions in P. Clearly, the two policies cannot be satisfied simultaneously. We formally define our notion of consistency amongst such policies in the following definition.

Definition 3.3.1 Given a set F of resiliency and separation of duty policies, the policies in F are *consistent* if and only if there exists an access control state UP such that UP satisfies every policy in F. Determining whether F is consistent is called the *Policy Consistency Checking Problem* (PCCP).

The following lemma asserts that the actual value of s and d in a resiliency does not affect its compatibility with SSoD policies. This enables us to replace all resiliency policies in the form of $\mathsf{rp}\langle P_i, s_i, d_i, t_i\rangle$ in F with the special form $\mathsf{rp}\langle P_i, 0, 1, t_i\rangle$ when studying $PCCP$. This greatly simplifies the problem.

Lemma 3.3.1 F is a set of policies and $R = \mathsf{rp}\langle P, s, d, t\rangle \in F$. Let $R' = \mathsf{rp}\langle P, 0, 1, t\rangle$ and $F' = (F - \{R\}) \cup \{R'\}$. F is consistent if and only if F' is consistent.

Proof It is clear that if F is consistent then F' is consistent. In the following, we prove that if F' is consistent then F is consistent. Assume that state UP' satisfies all policies in F'. UP' satisfying R' implies that there is a set U of no more than t users together have all the permissions in P. We then construct a new state UP by adding $s + d - 1$ copies of all users in U to UP'. Note that adding copies of existing users in UP' will not lead to violation of SSoD policies in F'. In this case, UP satisfies R plus all policies in F'. In other words, UP satisfies all policies in F and F is consistent. ■

The following theorem gives the computational complexity results about general cases of PCCP. Observe that the case with one SSoD policy and an arbitrary number of resiliency policies is $\mathbf{coNP}$-hard, and the case with one resiliency policy and an arbitrary number of SSoD policies is $\mathbf{NP}$-hard. Therefore, it is unlikely that the general case is in $\mathbf{NP}$ or in $\mathbf{coNP}$; however, we show that the problem is in $\mathbf{NP^{NP}}$.

Theorem 3.3.2 The computational complexities for PCCP are as follows:

1. PCCP $\langle 1, n\rangle$ is $\mathbf{coNP}$-hard, where PCCP $\langle 1, n\rangle$ denotes the subcase that there is a single SSoD policy, and an arbitrary number of resiliency policies.

2. PCCP $\langle m, 1\rangle$ is $\mathbf{NP}$-hard, where PCCP $\langle m, 1\rangle$ denotes the subcase that there is an arbitrary number of SSoD policies, and a single resiliency policy.

3. PCCP $\langle m, n\rangle$, i.e., the most general case of PCCP, is in $\mathbf{NP^{NP}}$.

We prove Theorem 3.3.2 by proving Lemmas 3.3.3, 3.3.4 and 3.3.5. Without loss of generality, we assume that for any static separation of duty policy $\mathsf{sod}\langle P, k\rangle$, we have $k \leq |P|$. We also assume that in any resiliency policy $\mathsf{rp}\langle P, s, d, t\rangle$, we have either $t = \infty$ or $t \leq |P|$.

Lemma 3.3.3 PCCP $\langle 1, n\rangle$ is $\mathbf{coNP}$-hard, where PCCP $\langle 1, n\rangle$ denotes the subcase that there is a single SSoD policy, and an arbitrary number of resiliency policies.

Proof We reduce the $\mathbf{NP}$-complete SET COVERING problem [24] (also referred to as MINIMUM COVERING problem in [25]) to the complement of $PCCP$. In SET COVERING, we are given a set $X = \{e_1, \cdots, e_m\}$, n subsets of X: $X_1, \ldots, X_n$, and a budget b, and need to determine whether the union of b subsets is the same as X. Given an instance of the SET COVERING problem, we construct one SSoD policy $S = \mathsf{sod}\langle P, b + 1\rangle$ and b rp policies $R_i = rp\langle P_i, 0, 1, 1\rangle$ $(1 \leq i \leq b)$, where $P = \{p_1, \cdots, p_m\}$ corresponds to X and $P_i = \{p_j \mid e_j \in X_i\}$ corresponds to X_i. Let $F = \{S, R_1, \cdots, R_n\}$. In the following, we prove that F is inconsistent if and only if the answer to the SET COVERING problem is "yes".

On the one hand, if F is inconsistent, there does not exist any state that satisfies all polices in F. In other words, if a state satisfies all resiliency policies in F, there exists no more than b users in the state who together have all the permission in P. Let UP be a state with n users $u_1, \cdots, u_n$ such that $(u_i, p_j) \in UP$ if and only if $p_j \in P_i$. It is clear that UP satisfies all resiliency policies in F, and hence there exist no more than b users together have all the permissions in P. In other words, there exist no more than b elements in $\{P_1, \cdots, P_n\}$ whose union is P. Thus, the answer to the set covering problem is "yes".

On the other hand, if the answer to the set covering problem is "yes", then there exist no more than b elements in $\{P_1, \cdots, P_n\}$ whose union is P. For any state UP that satisfies all resiliency policies in F, let U be the set of users that satisfy at least one resiliency policy. $u \in U$ if and only if there exists P_i such that u has all permissions in P_i. In this case, there exist no more than b users in U who together have all the permissions in P. Hence, UP does not satisfy S, which implies that no state satisfies all policies in F. ■

Lemma 3.3.4 PCCP $\langle m, 1\rangle$ is **NP**-hard, where PCCP $\langle m, 1\rangle$ denotes the subcase that there is an arbitrary number of SSoD policies, and a single resiliency policy.

Proof We reduce the **NP**-complete SET SPLITTING problem to $PCCP$. In the SET SPLITTING problem, we are given a set $X = \{e_1, \cdots, e_n\}$, m subsets of X: $X_1, \ldots, X_m$, and need to determine whether there exist Y_1 and Y_2 such that $Y_1 \cup Y_2 = X$ and there does not exist X_i $(1 \leq i \leq m)$ such that $X_i \subseteq Y_1$ or $X_i \subseteq Y_2$. Given an instance of the SET SPLITTING problem, construct a resiliency policy $R = \mathsf{rp}\langle P, 0, 1, 2\rangle$ and m SSoD policies $S_i = \mathsf{sod}\langle P_i, 2\rangle$ $(1 \leq i \leq m)$, where $P = \{p_1, \cdots, p_n\}$ corresponds to X and $P_i = \{p_j \mid e_j \in X_i\}$ corresponds to X_i. Let $F = \{R, S_1, \cdots, S_m\}$. In the following, we prove that F is consistent if and only if the answer to the SET SPLITTING problem is "yes".

On the one hand, if F is consistent, then there exists a state UP that satisfies all policies in F. UP satisfying R implies that there exist two users u_1 and u_2 in UP such that u_1 and u_2 together have all the permissions in P. Furthermore, UP satisfying S_i implies that neither u_1 nor u_2 has all permissions in P_i. Let $Y_1 = \{e_i \mid (u_1, p_i) \in UP\}$ and $Y_2 = \{e_i \mid (u_2, p_i) \in UP\}$. We have $Y_1 \cup Y_2 = X$ and neither Y_1 nor Y_2 is a superset of any X_i. The answer to the set splitting problem is "yes".

On the other hand, if the answer to the set splitting problem is "yes", then such Y_1 and Y_2 exist. We construct a state UP containing only two users u_1 and u_2 such that $(u_i, p_j) \in UP$ $(1 \leq i \leq 2)$ if and only if $p_j \in Y_i$. Since $Y_1 \cup Y_2 = X$, u_1 and u_2 together have all the permissions in P. Furthermore, since there does not exist X_i such that X_i is a subset of Y_1 or Y_2, neither u_1 nor u_2 has all permissions in P_i, which implies that UP satisfies S_i. Therefore, UP satisfies all policies in F. ■

Lemma 3.3.5 Let $F = \{S_1, S_2, \cdots S_m, R_1, \cdots, R_n\}$, where $S_i = \mathsf{sod}\langle P_i, k_i\rangle$ $(1 \leq i \leq m)$ and $R_j = \mathsf{rp}\langle Q_j, s_j, d_j, t_j\rangle$ $(1 \leq j \leq n)$. Checking whether policies in F are consistent is in $\mathbf{NP^{NP}}$.

Proof We construct a set of policies F' by replacing every R_i $(1 \leq i \leq n)$ in F with $\mathsf{rp}\langle P_i, 0, 1, t_i\rangle$. From Lemma 3.3.1, F is consistent if and only if F' is consistent.

We construct a nondeterministic Oracle Turing machine M that makes use of an $\mathbf{NP}$ oracle machine to determine whether F' is consistent. M first nondeterministically selects an integer a such that $max(k_1, \cdots, k_m) \leq a \leq \Sigma_{i=1}^{n}|Q_i|$ and then generates a users. Note that at least $max(k_1, \cdots, k_m)$ users are needed to satisfy all SSoD policies in F', and at most $\Sigma_{i=1}^{n}|Q_i|$ users are needed to satisfy all resiliency policies in F'. (The state can have more than $\Sigma_{i=1}^{n}|Q_i|$ users, but in order to show that all resiliency policies in F' are satisfied, at most $\Sigma_{i=1}^{n}|Q_i|$ users need to be involved.) Then M constructs a state UP by nondeterministically assigning a subset of Q to u, where $Q = \bigcup_{i=1}^{n} Q_i$ is the set of all permissions that appear in the resiliency policies. Next, M nondeterministically construct n sets $U_1, \cdots, U_n$ of users in UP, and then, for every $i \in [1, n]$, checks whether users in U_i together have all the permissions in P_i and $|U_i| \leq t_i$. If the answer is "no", then M returns `False`. Finally, M invokes the $\mathbf{NP}$ oracle to check whether UP violates any SSoD policy. (In order to prove that a state violates a static separation of duty policy $\mathsf{sod}\langle P, k\rangle$, we just need to present a set of no more than k users in the state who together have all the permissions in P. Therefore, checking whether a state violates an SSoD policy is in $\mathbf{NP}$.) If the oracle machine answers "yes", M returns `False`. Otherwise, M returns `True`, which means that UP satisfies all policies in F' and hence F' is consistent. It is clear that

M terminates in polynomial time if the oracle machine returns an answer instantaneously. Therefore, PCCP is in $\mathbf{NP}^{\mathbf{NP}}$ in general. ■

4 SATISFIABILITY AND RESILIENCY IN WORKFLOW AUTHORIZATION SYSTEMS

In the last two chapters, we have discussed high-level access control policies, such as policies specified by the algebra and resiliency policies, that apply to general access control systems. In this chapter, we study a lower-level access control scheme, the workflow authorization system.

Workflows are used in numerous domains, including production, purchase order processing, and various management tasks. A workflow divides a task into a set of well-defined sub-tasks (called *steps* here). Workflow authorization systems manage access control in workflows and have gained popularity in the research community [4–8]. As stated in Chapter 1, security policies in workflow authorization systems are usually specified using authorization constraints. One may specify, for each step, which users are authorized to perform it. In addition, one may specify the constraints between users who perform different steps in the workflow. For example, one may require that two steps must be performed by different users for the purpose of separation of duty [2]. Oftentimes, constraints in workflow authorization systems need to refer to relationships among users. For example, the rationale under a separation of duty policy that requires 2 users to perform the task is that this deters and controls fraud, as the collusion of 2 users are required for a fraud to occur. However, when two users are close relatives, then collusion is much more likely. To achieve the objective of deterring and controlling fraud, the policy should require that two different steps in a workflow must be performed by users who are not in conflict of interest with each other. In different environments, the conflict-of-interest relation need to be defined differently. For instance, inside an organization's system, relationships such as close relatives (e.g., spouses and parent-child) can be maintained and users who are close relatives may be considered to be in conflict of interest. In a peer-review setting, conflict of interest may be based on past collaborations, common institutions, etc. For another ex-

ample on user relations, one university may have a policy that a graduate student's study plan must be first approved by the student's advisor and then by the graduate officer in the student's department. To specify such constraints, one needs to define and refer to the advisor-student binary relation as well as the in-the-same-department binary relation.

In this chapter, we introduce the role-and-relation-based access control (R^2BAC) model for workflow authorization systems. The model is role-based in the sense that individual steps of a workflow are authorized to roles. The model is relation-based in the sense that user-defined binary relations can be used to specify constraints and an authorized user is prevented from performing a step unless the user satisfies these constraints. R^2BAC is a natural step beyond Role-Based Access Control (RBAC) [18], especially in the setting of workflows. As a role defines a set of users, which can be viewed as a unary relation among the set of all users, a binary relation is the natural next step.

A fundamental problem in workflow authorization systems is the *workflow satisfiability problem* (WSP), which asks whether a workflow can be completed in a certain access control state. We show that WSP is $\mathbf{NP}$-complete in R^2BAC. Furthermore, we show that the intractability is inherent in any workflow authorization systems that support some simple kinds of constraints. In particular, we show that WSP is $\mathbf{NP}$-hard in any workflow system that supports *either* constraints that require two steps must be performed by different users *or* constraints that require one step must be performed by a user who also performs at least one of several other steps. Such intractability results are somewhat surprising and discouraging, because the constraints involved are simple and natural. It is also unsatisfying as such results do not shed light on the computation cost one has to pay to enhance expressive power by introducing user-defined binary relations such as conflict-of-interest relation, since WSP is $\mathbf{NP}$-complete with or without user-defined relations. Finally, the practical significance of such intractability results is unclear, as in real world, certain aspects such as the number of steps in a workflow should be small. To address these issues, we apply tools from *parameterized complexity* [27] to WSP. Parameterized complexity is a measure of computational complexity of problems with multiple input parameters. Parameterized

complexity enables us to perform finer-grained study on the computational complexity of WSP.

In many situations, it is not enough to ensure that a workflow can be completed in the current access control state. In particular, when the workflow is designed to complete a critical task, it is necessary to make sure that the workflow can be completed even if certain users become absent in emergency situations. In other words, *resiliency* is important in workflow authorization systems. Resiliency in workflow authorization systems differs from the resiliency policies proposed in Chapter 3 in two major aspects. First, due to the existence of authorization constraints, even if a set of users together are authorized to perform all steps in a workflow, it is still possible that they cannot complete the task. Second, as a workflow consists of a sequence of steps and finishing all these steps may take a relatively long time, it is possible that certain users become absent at some point and come back later. In other words, the set of available users may change during the execution of a workflow. Therefore, more refined notions of resiliency for workflow authorization systems are needed. In this chapter, we introduce three levels of resiliency in workflow authorization systems and study the complexity of resiliency checking.

The remainder of this chapter is organized as follows. We introduce the R^2BAC model in Section 4.1. After that, we study the workflow satisfiability problem in Section 4.2 and study parameterized complexity of the problem in Section 4.3. Finally, we define and study resiliency problems in workflow systems in Section 4.4.

4.1 The Role-and-Relation-Based Access Control Model for Workflow Systems

In this section, we introduce the Role-and-Relation-Based Access Control (R^2BAC) model for workflow systems. We start with a motivating example.

Example 4 In an academic institution, submitting a grant proposal to an outside sponsor via the sponsor program services (SPS) is modeled as a workflow with five steps[1] (see Figure 4.1).

[1]This is a simplified version of the process in the authors' institution, which also requires signatures of the department head and the dean's office.

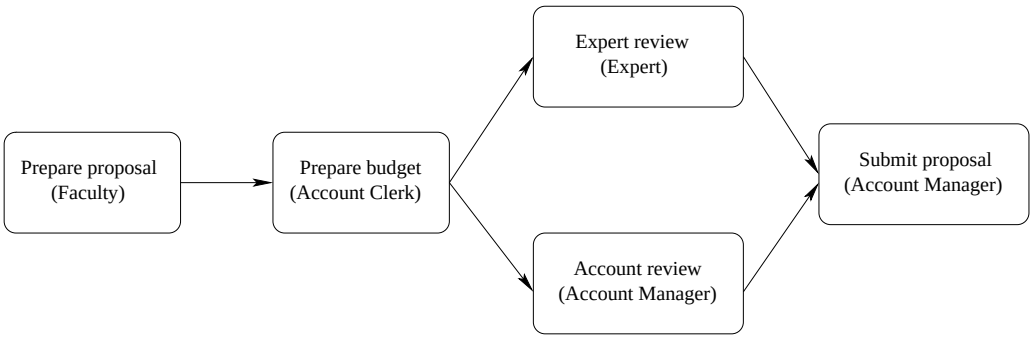


Figure 4.1. A workflow for grant proposal submission to outside sponsor via the sponsor program services (SPS).

1. `Preparation`: A faculty member prepares a proposal and sends it to the business office of his or her department.

2. `Budget`: An account clerk prepares the budget, checks the proposal, and submits it to the SPS office.

3. `Expert Review`: A regulation expert in the SPS office reviews the proposal to check whether the proposal satisfies various regulations, e.g., those governing export control and human subject research.

4. `Account Review`: An account manager reviews the proposal and the budget.

5. `Submission`: An account manager submits the proposal to the outside sponsor.

In the workflow, steps `expert review` and `account review` may be performed concurrently while all other steps must be carried out sequentially. The step `preparation` can be performed by any personnel who can serve as a primary investigator, while the step `budget` must be carried out by an account clerk. A regulation expert is authorized to review the proposal in the step `expert review`. The privilege to perform steps `account review` and `submission` is granted to account managers.

The workflow has the following constraints.

1. Steps `preparation`, `budget`, `expert review` and `account review` must be performed by four different users.

2. The account clerk who signs the proposal must be in the same department as the faculty member who prepares the proposal.

3. The persons who review the proposal must not have a conflict of interest with the one submitting the proposal.

4. The account manager who reviews the proposal is responsible to submit it to the outside sponsor.

In the above, Constraint 2 reflects certain procedural and duty requirements, while Constraint 1 enforces the principle of separation of duty. Constraint 3 follows the spirit of separation of duty and goes beyond that. Rather than simply requiring that the two steps must be performed by different people, the constraint requires that the people who perform the two steps must not have a conflict of interest. Constraint 4 enforces a binding-of-duty policy [6] by requiring two tasks be performed by the same user.

As security and practical requirements vary from tasks to tasks, specification of constraints plays a crucial role in expression of workflows. As demonstrated in Example 4, binary relations play an important role in expressing authorization constraints. Most existing workflow authorization models support only a few pre-defined binary relations, which limits the expressive power of these models. For example, the model proposed in [7] supports only six pre-defined binary relations $\{=, \neq, <, \leq, >, \geq\}$ between users and roles. Hence, there is no way to express relations like "in the same department" or "is a family member". The model in [6] supports user-defined relations. Our role-and-relation-based access control (R^2BAC) model for workflow systems extends the model in [6] by explicitly combining roles and relations and by supporting more sophisticated forms of constraints using these relations.

We now introduce formal definitions for R^2BAC. Note that $\mathcal{U}$, $\mathcal{R}$ and $\mathcal{B}$ are names of all possible users, roles and binary relations in the system, respectively.

Definition 4.1.1 (Access Control State) An *access control state* is given by a tuple $\langle U, UR, B\rangle$, where $U \subseteq \mathcal{U}$ is a set of users, $UR \subseteq U \times \mathcal{R}$ is the user-role membership relation and $B = \{\rho_1, \cdots, \rho_m\} \subseteq \mathcal{B}$ is a set of binary relations such that $\rho_i \subseteq U \times U$ $(i \in [1, m])$. For convenience, we assume that when ρ is in B, $\overline{\rho}$ is also in B, and $(u_1, u_2) \in \overline{\rho}$ if and only if $(u_1, u_2) \notin \rho$. Also, $\overline{\overline{\rho}}$ is the same as ρ. Furthermore, we assume that B contains two predefined binary relations "$=$" and "$\neq$", which denote equality and inequality, respectively.

An access control state $\langle U, UR, B\rangle$ defines the environment in which a workflow is to be run. In particular, B should define all the binary relations that appear in any constraint in workflows to be run in the environment.

Definition 4.1.2 (Workflow and Constraints) A *workflow* is represented as a tuple $\langle S, \preceq, SA, C\rangle$, where S is a set of steps, $\preceq \subseteq S \times S$ defines a partial order among steps in S, $SA \subseteq \mathcal{R} \times S$, and C is a set of constraints, each of which takes one of the following forms:

1. $\langle \rho(s_1, s_2)\rangle$: the user who performs s_1 and the user who perform s_2 must satisfy the binary relation ρ.

2. $\langle \rho(\exists X, s)\rangle$: there exists a step $s' \in X$ such that $\langle \rho(s', s)\rangle$ holds, i.e., the user who performs s' and the user who performs s satisfy ρ.

3. $\langle \rho(s, \exists X)\rangle$: there exists a step $s' \in X$ such that $\langle \rho(s, s')\rangle$ holds.

Intuitively, in a workflow $\langle S, \preceq, SA, C\rangle$, that $s_i \preceq s_j$ $(i \neq j)$ indicates that step s_i must be performed before step s_j. Steps s_i and s_j may be performed *concurrently*, if neither $s_i \preceq s_j$ nor $s_j \preceq s_i$. SA is called *role-step authorization* and $(r, s) \in SA$ indicates that members of role r are authorized to perform step s.

Also, we may introduce a new type of constraint $\langle \rho(\forall X)\rangle$ to require that for any two steps $s_i, s_j \in X$, $\langle \rho(s_i, s_j)\rangle$ must hold. Such a constraint can be equivalently represented using $|X|^2$ constraints in the form of $\langle \rho(s_1, s_2)\rangle$. The new constraint is a syntactic sugar when we would like to express a requirement that users who perform certain steps must

have relation ρ with each other. For instance, the constraint $\langle \neq (\forall X) \rangle$ states that no user may perform more than one steps in X. Similarly, we may define another syntactic sugar $\langle \rho(s, \forall X) \rangle$ which requires that the user who performs s has relation ρ with every user who performs a step in X.

Example 5 Consider the workflow for submitting a grant proposal in Example 4. Let $s_{prepare}$, s_{budget}, s_{xp_review}, s_{ac_review} and s_{submit} denote the five steps in the workflow. The constraints of the workflow can be represented in tuple-based specification as follows.

1. $\langle \neq (\forall \{s_{prepare}, s_{budget}, s_{xp\_review}, s_{ac\_review}\}) \rangle$

   These require that the first four steps in the workflow must be performed by four different users.

2. $\langle \rho_{same\_dept}(s_{budget}, s_{prepare}) \rangle$

   $(u_x, u_y) \in \rho_{same\_dept}$ when u_x and u_y are in the same department. The constraint requires that the person who signs the proposal must be in the same department as the person who prepares it.

3. $\langle \overline{\rho}_{conflict\_interest}(s_{xp\_review}, s_{prepare}) \rangle$
   $\langle \overline{\rho}_{conflict\_interest}(s_{ac\_review}, s_{prepare}) \rangle$

   $(u_x, u_y) \in \rho_{conflict\_interest}$ when u_x and u_y have a conflict of interest. The constraint requires that the person who reviews the proposal must not have a conflict of interest with the person who prepares it.

4. $\langle = (s_{submit}, s_{ac\_review}) \rangle$

   The constraint requires that `account review` and `submission` must be performed by the same person.

Definition 4.1.3 (Plans and Partial Plans) A *plan* P for workflow $W = \langle S, \preceq, SA, C \rangle$ is a subset of $\mathcal{U} \times S$ such that, for every step $s_i \in S$, there is exactly one tuple (u_a, s_i) in P, where $u_a \in \mathcal{U}$.

A *partial plan* PP for W is a subset of $\mathcal{U} \times S$ such that, for every step $s_i \in S$, there is at most one tuple (u_a, s_i) in PP, where $u_a \in \mathcal{U}$. And $(u_a, s_i) \in PP$ implies that, for every $s_j \preceq s_i$, there exists $u_b \in \mathcal{U}$ such that $(u_b, s_j) \in PP$.

Intuitively, a plan assigns exactly one user to every step in a workflow, while a partial plan does this for only a portion of the steps in the workflow. Furthermore, if a step is in a partial plan, then its prerequisite steps must also be in the partial plan.

Definition 4.1.4 (Valid Plan) Given a workflow $W = \langle S, \preceq, SA, C \rangle$, and an access control state $\gamma = \langle U, UR, B \rangle$, we say that a user u is an *authorized user* of a step $s \in S$ under γ if and only if there exists a role r such that $(u, r) \in UR$ and $(r, s) \in SA$.

We say that a plan P *is valid for* W under γ if and only if for every $(u, s) \in P$, u is an authorized user of s, and no constraint in C is violated. We say that W is *satisfiable* under γ if and only if there exists a plan P that is valid for W under γ.

Note that there can be multiple valid plans for a workflow W in an access control state. In fact, it is the existence of multiple valid plans that makes it possible for W to be completed even if a number of users are absent. In situations where the access control state changes during the execution of a workflow instance (e.g. users become absent), we will have to change our plan at runtime and thus constraints need to be checked at runtime as well. Constraints are checked before the last step restricted by the constraint is to be executed.

Definition 4.1.5 (Valid Partial Plan) Given a workflow $\langle S, \preceq, SA, C \rangle$ and an access control state $\langle U, UR, B \rangle$, let $s_1, \cdots, s_m$ be a sequence of steps such that $s_i \not\preceq s_j$ when $i > j$. A partial plan PP is *valid* with respect to the sequence $s_1, \cdots, s_i$ if it assigns one user to each step in $s_1, \cdots, s_i$ and no constraint that is checked before the execution of s_i is violated by PP.

4.2 The Workflow Satisfiability Problem

A fundamental problem in workflow authorization systems is the Workflow Satisfiability Problem (WSP), which checks whether a workflow W is satisfiable under an access control state γ. Note that, given an access control state $\langle U, UR, B\rangle$, checking whether W is satisfiable under γ is equivalent to checking whether there is a valid plan for W under γ. In this section, we study the computational complexity of WSP.

4.2.1 Computational Complexity of WSP for R^{2}BAC

Theorem 4.2.1 WSP is $\mathbf{NP}$-complete in R^{2}BAC.

The proof of Theorem 4.2.1 consists of two parts. The first part is Lemma 4.2.2, which shows that WSP is in $\mathbf{NP}$ in R^{2}BAC. In the second part, Lemma 4.2.3 and Lemma 4.2.4 show that WSP is $\mathbf{NP}$-hard in two restricted cases.

Lemma 4.2.2 WSP is in $\mathbf{NP}$ in R^{2}BAC.

Proof The length of a plan is bounded by the number of steps in the workflow. Given a plan for a workflow, checking whether a user is authorized to perform a step can be done in linear time. Also, checking whether a constraint is satisfied by the plan can be done in polynomial time. Hence, checking whether a plan is valid can be done in polynomial time. A nondeterministic Turing machine can thus guess a plan and check whether it is valid in polynomial time. ■

Lemma 4.2.3 WSP is $\mathbf{NP}$-hard in R^{2}BAC, if the workflow uses constraints of the form $\langle \neq (s_1, s_2)\rangle$.

Please refer to Appendix B.1 for the proof of the above lemma. In the proof, we use a reduction from the $\mathbf{NP}$-complete GRAPH K-COLORABILITY problem. In the reduction, vertices in a graph are mapped to steps in the workflow, while colors are mapped to users. In the GRAPH K-COLORABILITY problem, the number of vertices is normally much larger

than the number of colors. Hence, the number of steps in the constructed workflow is much larger than the number of users, which is rarely the case in practice. Such a phenomenon indicates that classical complexity framework is inadequate to study the complexity of WSP in a real-word setting. This motivates us to apply the tool of parameterized complexity to perform finer-grained study of the complexity of WSP, which will be discussed in Section 4.3.

Lemma 4.2.4 WSP is $\mathbf{NP}$-hard in R^{2}BAC, if the workflow uses constraints of the form $\langle = (s, \exists X)\rangle$.

Please refer to Appendix B.1 for the proof of the above lemma. In the proof, we use a reduction from the $\mathbf{NP}$-complete HITTING SET problem.

Although WSP is intractable in general in R^{2}BAC, the problem is in $\mathbf{P}$ for certain special cases. Lemma 4.2.5 states a tractable case of WSP.

Lemma 4.2.5 WSP is in $\mathbf{P}$ in R^{2}BAC, if the workflow only has constraints in the forms of $\langle = (s_1, s_2)\rangle$.

Proof Given a step s, let $AU(s)$ be the set of users authorized to perform step s. A constraint $\langle = (s_1, s_2)\rangle$ requires s_1 and s_2 be performed by the same user. In this case, if $AU(s_1) \cap AU(s_2) = \emptyset$, then it is impossible to perform the two steps without violating the constraint and the answer to the WSP instance is "no". Otherwise, we replace $AU(s_1)$ and $AU(s_2)$ with $AU(s_1) \cap AU(s_2)$ and then repeat the process for another constraint in the workflow until all constraints in the workflow have been processed. If we finish processing all constraints without answering "no", the answer to the WSP instance is "yes". Since set intersection can be done in polynomial time and we need to compute at most $|C|$ intersections, a given WSP instance can be answered in polynomial time. ■

4.2.2 The Inherent Complexity of Workflow Systems

In Section 4.2.1, we show that WSP is $\mathbf{NP}$-hard in R^{2}BAC in general. In this section, we point out that the intractability of WSP is inherent to certain fundamental features of

workflow authorization systems and is independent of modeling approaches. We say that a workflow system supports the feature of *user-step authorization* if it allows one to specify (either directly or indirectly) which users are allowed to perform which steps in the workflow. User-step authorization is probably the most fundamental feature and almost all workflow systems found in existing literature support such a feature. A *user-inequality constraint* states that certain two steps cannot be performed by the same user, i.e., $\langle \neq (s_1, s_2) \rangle$ in R^{2}BAC. An *existence-equality constraint* states that a certain step must be performed by a user who performs at least one step in a given set of steps, i.e., $\langle = (s, \exists X) \rangle$ in R^{2}BAC.

Theorem 4.2.6 Checking whether a set of users can complete a workflow is **NP**-hard for any workflow system that supports user-step authorization and user-inequality constraints.

Proof The reduction from GRAPH K-COLORABILITY in the proof of Lemma 4.2.3 only makes use of user-step authorization and user-inequality constraints offered by R^{2}BAC. Therefore, the reduction also applies to the satisfiability problem for any workflow system that supports these two features. ■

Note that user-inequality constraints are widely used in existing literature to enforce separation of duty in workflow systems. Many workflow models [5–7] support such type of constraints.

Theorem 4.2.7 Checking whether a set of users can complete a workflow is **NP**-hard for any workflow system that supports user-step authorization and existence-equality constraints.

Proof The reduction from HITTING SET in the proof of Lemma 4.2.4 only makes use of user-step authorization and existence-equality constraints offered by R^{2}BAC. Therefore, the reduction also applies to the satisfiability problem for any workflow system that supports these two features. ■

Note that existence-equality constraint is a natural way to enforce the general form of binding of duty policies, which require a step be performed by one of those users who have performed some prerequisite steps.

4.3 Beyond Intractability of WSP

In Section 4.2, we have shown that WSP is NP-complete in R^2BAC for the general case as well as the two special cases where only a simple form of constraints are used. Such results, however, are unsatisfying, as they do not shed light on the computation cost associated with introducing additional expressive features such as user-defined binary relations, since the complexity of WSP is NP-complete in all the three cases. Such a phenomenon indicates that classical computational complexity does not precisely capture the computational difficulty of different cases of WSP. Furthermore, the practical significance of such intractability results is unclear. The input to WSP consists of many aspects, such as the number of steps in the workflow, the number of constraints and the number of users in the access control state etc. In practice, some aspects of the input do not take a large value. For instance, even though the number of users may be large, the number of steps in the workflow is expected to be small. An interesting question that arises is whether WSP can be solved efficiently given the restriction that the number of steps is small.

To address these issues, we apply tools from the theory of parameterized complexity [27] to WSP.

4.3.1 Why Parameterized Complexity?

Parameterized complexity is a measure of complexity of problems with multiple input parameters. The theory of parameterized complexity was developed in the 1990s by Rod Downey and Michael Fellows. It is motivated, among other things, by the observation that there exist hard problems that (most likely) require exponential runtime when complexity is measured in terms of the input size only, but that are computable in a time that is polynomial in the input size and exponential in a (small) parameter k. Hence, if k is fixed at a small value, such problems can still be considered 'tractable' despite their traditional classification as 'intractable'.

In classical complexity, a decision problem is specified by two items of information: (1) the input to the problem, and (2) the question to be answered. In parameterized complexity,

there are three parts of a problem specification: (1) the input to the problem, (2) the aspects of the input that constitute the parameter, and (3) the question to be answered. Normally, the parameter is selected because it is likely to be confined to a small range in practice. The parameter provides a systematic way of specifying restrictions of the input instances. Some **NP**-hard problems can be solved by algorithms that are exponential only in a fixed parameter while polynomial in the size of the input. Such an algorithm is called a *fixed-parameter tractable* algorithm. More specifically, an algorithm for solving a problem is a fixed-parameter tractable algorithm, if when given any input instance of the problem with parameter k, the algorithm takes time $O(f(k)n^{\alpha})$, where n is the size of the input, k is the parameter, α is a constant (independent of k), and f is an arbitrary function.

If a problem has a fixed-parameter tractable algorithm, then we say that it is a fixed-parameter tractable problem and belongs to the class **FPT**. For example, the **NP**-complete VERTEX COVER asks, given a graph G and an integer k, whether there is a size-k set V' of vertices, such that every edge in G is adjacent to at least one vertex in V'. This problem is in **FPT** when taking k as the parameter, as there exists a simple algorithm with running time of $O(2^k n)$, where n is the size of G. Note that not all intractable problems are in **FPT**. For instance, the **NP**-complete DOMINATING SET problem is fixed-parameter intractable. Given a graph G and an integer k, DOMINATING SET asks whether there is a size-k set V' of vertices such that every vertex in G is either in V' or is connected to a vertex in V' by an edge. For DOMINATING SET, there is no significant alternative to trying all size-k subsets of vertices in G and there are $O(n^k)$ such subsets, where n is the number of vertices.

Finally, we would like to point out that a problem in **FPT** does not necessarily mean that it can be efficiently solved as long as the parameter is small. Note that $f(k)$ may be a function that grows very fast over k. For instance, an $O(k^{k^k} n)$ algorithm is not practical even if k is as small as 5, just as we cannot claim that a problem in **P** can be solved efficiently when the best algorithm takes time $O(n^{100})$. However, showing that a problem is in **FPT** has significant impact as experiences have shown that improvement on fixed-parameter tractable algorithms are oftentimes possible. For instance, when VERTEX COVER was first observed to be solvable in $O(f(k)n^3)$, $f(k)$ was such a function that the

algorithm is utterly impractical even for $k = 1$. An $O(2^k n)$ algorithm was proposed later, and then an algorithm with running time $O(kn + (4/3)^k k^2)$ was revealed. Right now, VERTEX COVER is well-solved for input of any size, as long as the parameter value is $k \leq 60$. Parameterized complexity offers a fresh angle into designing algorithms for such problems.

In this dissertation, we only study which subcases of WSP are in $\mathbf{FPT}$ and which are not. Improvement on the fixed-parameter tractable algorithms for the $\mathbf{FPT}$ cases is beyond the scope of this dissertation.

4.3.2 Fixed Parameter Tractable Cases of WSP

As the number of steps in a workflow is likely to be small in practice, we select the number of steps as the parameter for WSP. We first show that a special case of WSP in which only the $\neq$ relation is allowed is in $\mathbf{FPT}$. The proof gives a fixed-parameter tractable algorithm and illustrates the intuition why this problem is in $\mathbf{FPT}$.

Lemma 4.3.1 WSP in R^{2}BAC is in $\mathbf{FPT}$, if $\neq$ is the only binary relation used by constraints in the workflow. In particular, given a workflow W and an access control state γ, WSP can be solved in time $O(k^{k+1}n)$, where k is the number of steps in W and n is the size of the entire input to the problem.

Proof A constraint using binary relation $\neq$ requires a certain step to be performed by a user who does not perform certain other step(s). Since there are k steps in W, if step s is authorized to no less than k users in U, then we can always find an authorized user of s, who is not assigned to any other steps in W. In other words, we only need to consider those steps that are authorized to less than k users in U, and there are at most k such steps. We construct partial plans for these steps by trying all combinations of authorized users and there are no more than k^k such combinations. Verifying whether a plan is valid can be done in $O(kn)$, as there are $O(n)$ constraints and each constraints restricts at most k steps. Therefore, checking whether U can complete W can be done in time $O(k^{k+1}n)$. ■

The following Theorem subsumes Lemma 4.3.1.

Theorem 4.3.2 WSP is in $\mathbf{FPT}$ in R^{2}BAC, if = and $\neq$ are the only binary relations used by constraints in the workflow.

Please refer to Appendix B.1 for the proof of Theorem 4.3.2. In the proof of Theorem 4.3.2, we focus on showing that the problem is in $\mathbf{FPT}$. Improving the fixed-parameter tractable algorithm in the proof is beyond the scope of this dissertation.

4.3.3 WSP Is Fixed Parameterized Intractable in General

A natural question to ask is whether WSP is still in $\mathbf{FPT}$ when user-defined binary relations are allowed in the workflow. We show that the answer is "no". Similar to proving a problem is intractable in classical complexity framework, we prove that a problem is fixed-parameter intractable by reducing another fixed-parameter intractable problem to the target problem. To preserve fixed-parameter tractability, we need to use a kind of reduction different from the classical ones used in $\mathbf{NP}$-completeness proofs. We say that L reduces to L' by a *fixed-parameter reduction* if given an instance $\langle x, k\rangle$ for L, one can compute an instance $\langle x' = g_1(\langle x, k\rangle), k' = g_2(k)\rangle$ in time $O(f(k)|x|^{\alpha})$ such that $\langle x, k\rangle \in L$ if and only if $\langle x', k'\rangle \in L'$, where g_1 and g_2 are two functions and α is a constant. Note that many classical reductions are not fixed-parameter reduction as they do not carry enough structure, which leads to lose of control for the parameter.

Under parameterized complexity, each problem falls somewhere in the hierarchy: $\mathbf{P} \subseteq \mathbf{FPT} \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P] \subseteq \mathbf{NP}$. If a problem is $W[1]$-hard, then it is believed to be fixed-parameter intractable. To understand the classes $W[t]$, we can start by viewing a 3CNF formula as a (boolean) decision circuit, consisting of one input for each variable and structurally a large *and* gate taking inputs from a number of small *or* gates. (Some wires in the circuit may include a negation.) The *or* gates are small in that each of them takes 3 inputs, and the *and* gate is large in that it takes an unbounded number of inputs. The *weft* of a decision circuit is the maximum number of large gates on any path from the input variable to the output line. The *weighted satisfiability* problem for

decision circuits asks whether a decision circuit has a weight k satisfying assignment (i.e., a satisfying assignment in which at most k variables are set to true). The class $W[t]$ includes all problems that are fixed parameter reducible to the weighted satisfiability problem for decision circuits of weft t.

The following theorem states that WSP is fixed-parameter intractable in R^{2}BAC when user-defined binary relations are allowed in the workflow. The proof of the theorem is in Appendix B.1.

Theorem 4.3.3 WSP is $W[1]$-hard in R^{2}BAC if user-defined binary relations are used in constraints.

We conclude from Theorem 4.3.2 and Theorem 4.3.3 that supporting user-defined binary relations introduces additional complexity to WSP in R^{2}BAC. Parameterized complexity reveals such a fact that is hidden by classical complexity framework and allows us to better understand the source of complexity of WSP in R^{2}BAC. We point out that a naive algorithm solving WSP for R^{2}BAC, which enumerates all possible plans and verifies each of them, takes time $O(kn^{k+1})$, which may be acceptable when k and n are small. We also note that it is possible to develop algorithms with heuristic optimizations that can solve WSP efficiently for practical instances; the study of such algorithms is beyond the scope of this dissertation.

Finally, we provide an upperbound of the complexity of WSP in R^{2}BAC in the parameterized complexity framework.

Theorem 4.3.4 WSP in R^{2}BAC is in $W[2]$.

Please refer to Appendix B.1 for the proof of Theorem 4.3.4. It remains open whether WSP is $W[1]$-complete or $W[2]$-complete.

4.4 Resiliency in Workflow Systems

We have studied the workflow satisfiability problem (WSP) in previous sections. In many situations, it is not enough to ensure that a workflow is satisfiable in the current

access control state. In particular, when the workflow is designed to complete a critical task, it is necessary to guarantee that even if certain users are absent unexpectedly, the workflow can still be completed. Resiliency is a property of those system configurations that can satisfy the workflow even with absence of some users.

In this section, we define and study resiliency in workflow systems. The workflow model we use is R^2BAC. Before giving formal definitions of resiliency in workflow systems, let's consider three scenarios.

1. The execution of instances of a workflow is done in a relatively short period of time, say within fifteen minutes. Although it is possible that certain users are absent before the execution of a workflow instance, it is unlikely that available users become absent during the execution of the workflow instance. In other words, the set of users who are available for a workflow instance is stable.

2. The execution of instances of a workflow takes a relatively long period of time, say within one day. Some users may not come to work on the day when a workflow instance is executed. Furthermore, some users may have to leave at some point (e.g. between the execution of two steps) before the workflow instance is completed and will not come back to work until the next day. In such a situation, the set of users available to the workflow instance becomes smaller and smaller over time. Such a scenario would also be possible in potentially hazardous situations such as battlefield and fire-fighting.

3. The execution of instances of a workflow takes a long period of time. For example, only a single step of the workflow is performed each day. Since the set of users who come to work may differ from day to day, the set of available users may differ from step to step.

We capture the above three scenarios by proposing three levels of resiliency in workflow systems. They are *static (level-1) resiliency*, *decremental (level-2) resiliency* and *dynamic (level-3) resiliency*. In static resiliency, a number of users are absent before the execution

of a workflow instance, while remaining users will not be absent during the execution; in decremental resiliency, users may be absent before or during the execution of a workflow instance, and absent users will not become available again; in dynamic resiliency, users may be absent before or during the execution of a workflow instance and absent users may become available again. In all cases, we assume that the number of absent users at any point is bounded by a parameter t. We now give formal definitions of the three levels of resiliency.

Definition 4.4.1 (Static Resiliency) Given a workflow W and an integer $t \geq 0$, an access control state $\langle U, UR, B \rangle$ is *statically resilient* for W up to t absent users if and only if for every size-t subset U' of U, W is satisfiable under $\langle (U - U'), UR, B \rangle$.

Intuitively, an access control state is statically resilient for a workflow if the workflow is still satisfiable after removing t users from the access control state.

Definition 4.4.2 (Decremental Resiliency) Given a workflow $W = \langle S, \preceq, SA, C \rangle$ and an integer t, an access control state $\langle U, UR, B \rangle$ is *decrementally resilient* for W up to t absent users, if and only if Player 1 can always win the following two-person game when playing optimally.

Initialization: $PP \leftarrow \emptyset$, $U_0 \leftarrow U$, $S_0 \leftarrow S$, $t_0 \leftarrow t$ and $i \leftarrow 1$.

Round i of the Game:

1. Player 2 selects a set U'_{i-1} such that $|U'_{i-1}| \leq t_{i-1}$.

   $U_i \leftarrow (U_{i-1} - U'_{i-1})$ and $t_i \leftarrow (t_{i-1} - |U'_{i-1}|)$.

2. Player 1 selects a step $s_{a_i} \in S_{i-1}$ such that $\forall s_b (s_b \prec s_{a_i} \Rightarrow s_b \notin S_{i-1})$.

   Player 1 selects a user $u \in U_i$.

   $PP \leftarrow PP \cup \{(u, s_{a_i})\}$ and
   $S_i \leftarrow (S_{i-1} - \{s_{a_i}\})$.

   If PP is not a valid partial plan with respect to the sequence $s_{a_1}, \cdots, s_{a_i}$, then Player 1 loses.

3. If $S_i = \emptyset$, then Player 1 wins; otherwise, let $i \leftarrow (i+1)$ and the game goes on to the next round.

In each round, Player 2 may remove a certain number of users and then Player 1 has to pick a remaining step that is ready to be performed and assign an available user to it. The total number of users Player 2 may remove throughout the game is bounded by t. An access control state is decrementally resilient for a workflow if there is always a way to complete the workflow no matter when and which users are removed, as long as the total number of absent users is bounded by t.

Also, in Definition 4.4.2, we assume that Player 1 plays optimally, which implies that in each round, Player 1 has to consider not only the next step but also all future steps.

Example 6 There is a workflow $W = \langle S, \preceq, SA, C\rangle$ and an access control state $\langle U, UR, B\rangle$, where $S = \{s_1, s_2\}$, $s_1 \preceq s_2$, $C = \{\langle \neq (s_1, s_2)\rangle\}$, $SA = \{(r_1, s_1), (r_2, s_2)\}$, and $UR = \{(Alice, r_1), (Alice, r_2), (Bob, r_1), (Carl, r_2)\}$. All users in $U = \{Alice, Bob, Carl\}$ are available before the execution of s_1. Consider the following two choices of user assignment for s_1.

1. $Alice$ is assigned to perform s_1: If $Carl$ becomes absent after the execution of s_1, then $Alice$ is the only user authorized to perform s_2. However, assigning $Alice$ to s_2 violates the constraint $\langle \neq (s_1, s_2)\rangle$. That is to say, the remaining users cannot complete the workflow.

2. Bob is assigned to perform s_1: In this case, no matter which single user becomes absent after the execution of s_1, we can always find an authorized user (either $Alice$ or $Carl$) to perform s_2 without violating the constraint $\langle \neq (s_1, s_2)\rangle$.

Thus it is clear that having Bob perform s_1 is a better choice than having $Alice$ with respect to resiliency. Actually, it can be proved that this access control state is decrementally resilient for W up to one absent user.

Definition 4.4.3 (Dynamic Resiliency) Given a workflow $W = \langle S, \preceq, SA, C\rangle$ and an integer t, an access control state $\langle U, UR, B\rangle$ is *dynamically resilient* for W up to t absent

users, if and only if Player 1 can always win the following two-person game when playing optimally.

Initialization: $PP \leftarrow \emptyset$, $S_0 \leftarrow S$ and $i \leftarrow 1$.

Round i of the Game:

1. Player 2 selects a set U'_{i-1} of up to t users.

   $U_i \leftarrow (U - U'_{i-1})$.

2. Player 1 selects a step $s_{a_i} \in S_{i-1}$ such that $\forall s_b (s_b \prec s_{a_i} \Rightarrow s_b \notin S_{i-1})$.

   Player 1 selects a user $u \in U_i$.

   $PP \leftarrow PP \cup \{(u, s_{a_i})\}$ and
   $S_i \leftarrow (S_{i-1} - \{s_{a_i}\})$.

   If PP is not a valid partial plan with respect to the sequence $s_{a_1}, \cdots, s_{a_i}$, then Player 1 loses.

3. If $S_i = \emptyset$, then Player 1 wins; otherwise, let $i \leftarrow (i + 1)$ and the game goes on to the next round.

Intuitively, Player 2 may temporarily remove up to t users from the access control state at the beginning of each round. Then, Player 1 has to select a remaining step that is ready to be performed and assign an available user to it. After that, the access control state is restored and the next round of the game starts.

The following theorem states a relationship among the three levels of resiliency in workflow systems: dynamic (level-3) resiliency is stronger than decremental (level-2) resiliency, which is in turn stronger than static (level-1) resiliency.

Theorem 4.4.1 Given a workflow W, an access control state γ and an integer t, the following are true.

- If γ is dynamically resilient for W up to t absent users, then it is also decrementally resilient for W up to t absent users.

- If γ is decrementally resilient for W up to t absent users, then it is also statically resilient for W up to t absent users.

But the reverse of either of the above statements is not true.

Proof The game defining dynamic resiliency allows Player 2 to play any strategy he/she can in the game defining decremental resiliency. The same relation holds between the game defining decremental resiliency and the definition of static resiliency. Therefore, the theorem holds. ■

4.4.1 Computational Complexities of Checking Resiliency

Theorem 4.4.2 Checking whether an access control state γ is statically resilient for a workflow W up to t users, which is called the *Static Resiliency Checking Problem* (SRCP), is $\mathbf{NP}$-hard and is in $\mathbf{coNP^{NP}}$.

Proof When $t = 0$, SRCP degenerates to WSP. Since WSP is $\mathbf{NP}$-complete, SRCP is $\mathbf{NP}$-hard.

Next, we prove that the problem is in $\mathbf{coNP^{NP}}$. From Lemma 4.2.2, checking whether a workflow is satisfiable under an access control state $\langle U, UR, B\rangle$ is in $\mathbf{NP}$. We now construct a nondeterministic oracle Turing machine M that decides the complement of the problem. Assume that M has access to an $\mathbf{NP}$ oracle N which checks whether a workflow is satisfiable under an access control state. M nondeterministically selects a set U' of t users and asks N whether the workflow is satisfiable under $\langle (U - U'), UR, B\rangle$. If the answer is "yes", M returns "no"; otherwise, M returns "yes". In this case, M returns "yes" if and only if the answer to the SRCP instance is "no". In general, SRCP is in $\mathbf{coNP^{NP}}$. ■

It remains open whether SRCP is $\mathbf{coNP^{NP}}$-complete or not. Readers who are familiar with computational complexity theory will recognize that $\mathbf{coNP^{NP}}$ is a complexity class in the Polynomial Hierarchy. Because the Polynomial Hierarchy collapses when $\mathbf{P} = \mathbf{NP}$, showing that an $\mathbf{NP}$-hard decision problem is in the Polynomial Hierarchy, although is not

equivalent to showing that the problem is $\mathbf{NP}$-complete, has the same consequence: the problem can be solved in polynomial time if and only if $\mathbf{P}$ =$\mathbf{NP}$.

Theorem 4.4.3 Checking whether an access control state γ is decremental resilient for a workflow W up to t users, which is called the *Decremental Resiliency Checking Problem* (CRCP), is $\mathbf{PSPACE}$-complete.

Theorem 4.4.4 Checking whether an access control state γ is dynamically resilient for a workflow W up to t users, which is called the *Dynamic Resiliency Checking Problem* (DRCP), is $\mathbf{PSPACE}$-complete.

Please refer to Appendix B.2 for proofs of Theorem 4.4.3 and 4.4.4. In the proofs, we reduce the $\mathbf{PSPACE}$-complete QUANTIFIED SATISFIABILITY problem to CRCP or DRCP. Intuitively, we use user-step assignments in workflow to simulate truth assignments for boolean variables.

Note that given a workflow $W = \langle S, \preceq, SA, C\rangle$, there may not exist an access control state that is decrementally or dynamically resilient for W even just up to one absent user, when the equality relation is used in constraints. For instance, assume that $S = \{s_1, s_2\}$, $s_1 \preceq s_2$ and $C = \{\langle = (s_1, s_2)\rangle\}$. Constraint $\langle = (s_1, s_2)\rangle$ requires s_1 and s_2 be performed by the same user. (Such constraints appear in [6] under the name binding-of-duty constraints.) If the user who executed s_1 becomes absent before the execution of s_2, then there is no way to finish the workflow without violating the constraint no matter which users remain available. This illustrate that bind-of-duty constraints can make it difficult to achieve decremental or dynamic resiliency. This problem can be addressed either by not using such constraints in settings where such resiliency is desirable, or by introducing mechanisms such as delegation.

5 DELEGATION IN WORKFLOW AUTHORIZATION SYSTEMS

In the last chapter, we have studied satisfiability and resiliency in workflow authorization systems. There are a couple of ways to enforce resiliency policies in workflow authorization systems. One approach is to have enough redundancy of human resources in the system configuration. However, this approach leads to high costs. An alternative approach is to use user-to-user delegation (or delegation for short). Delegation is a mechanism that allows a user A to act on another user B's behalf by making B's access rights available to A. It is well recognized as an important mechanism to provide fault-tolerance and flexibility in access control systems, and has gained popularity in the research community [9–17].

Essentially, a delegation operation temporarily changes the access control state so as to allow a user to use another user's access privileges. While delegation can make an access control system more resilient to absence of users, it may lead to violation of security policies, especially static separation of duty policies. For instance, if role r_1 and role r_2 are mutually exclusive, then a user who is a member of r_1 should not be allowed to receive r_2 from others through delegation. In contrast to normal access right administration operations, which are performed centrally, delegation operations are usually performed in a distributed manner. That is to say, users have certain control on the delegation of their own rights. In order to prevent abuse, some delegation models support specification of authorization rules, which control who can delegate what privileges to other users as well as who can receive what privileges from others.

Delegation may be viewed as a module that introduces additional functionalities into access control systems. To enhance existing access control systems with delegation, one needs to incorporate a delegation module into those systems. A naive approach is to place the delegation module on top of the access control module, and let the delegation module handle delegation operations and manipulate access control configuration. For example, when $Alice$ delegates the role r to Bob, the access control configuration is modified so that

Bob is authorized for r in the new configuration. The underlying access control module consults the access control configuration without concerning delegation. Even though such a naive approach is simple and allows reusing existing implementation of access control modules, it introduces security breaches into the system. As we point out in Section 5.1.1, colluding users could exploit such breaches to circumvent security policies in the access control system. Due to the decentralized nature of delegation and the fact that not all the users in the system are trusted, collusion is a threat that must not be overlooked.

Since the naive approach could be insecure, more sophisticated methods are needed to create a secure system with delegation support. Surprisingly, even though delegation is well recognized as a very useful component of access control systems, to our knowledge, no work has performed in-depth study on how to incorporate a delegation module into access control systems in a secure manner. In this chapter, we formally define the notion of security with respect to delegation. Intuitively, if an access control system is secure, then any group of users cannot "enhance the power" (i.e. become capable to complete more tasks than before) of the group through mutual delegation within the group. To justify this intuition, by delegating her privileges to user A, user B allows A to work on her behalf. This indicates that A gains no more than what B has, and thus, A should not be able to do more than A and B together can do before the delegation operation. This further implies that, after the delegation operation, A and B as a group cannot do more than before. If a system does not have such a property, when A and B collude, they may gain extra power by delegating privileges to each other. In that case, a group of colluding users can do more than they are supposed to do with the "help" of delegation, and the system is thus considered to be insecure with respect to delegation.

The remainder of this chapter is organized as follows. We first provide a formal definition of security with respect to delegation in Section 5.1 and then study enforcement mechanisms for delegation security in Section 5.2. Finally, we discuss how to use delegation to enforce resiliency in Section 5.3.

5.1 Delegation in Workflow Authorization Systems

Delegation is a mechanism that allows a user A to act on another user B's behalf by making B's access rights available to A. Delegation is an effective approach to enforce resiliency in workflow authorization systems. To achieve resiliency, we may delegate the privileges of absent users to available users so that steps that are only authorized to absent users can now be completed. We will discuss how to enforce resiliency using delegation in detail in Section 5.3.

Even though delegation is effective in achieving resiliency, simply adding delegation support to a workflow authorization system may lead to security breaches. In particular, colluding users may circumvent security constrains in workflows using delegation. In this section, we study the impact of delegation on the security of workflow authorization systems. We first provide definitions related to delegation and formalize delegation operations as access control state transition operations. Next, in Section 5.1.1, we give examples on delegation-based attacks on workflow authorization systems. Finally, in Section 5.1.2, we formally define the notion of security with respect to delegation in access control systems.

Definition 5.1.1 (Access Control State with Delegation) An access control state γ is given as a tuple $\langle UR, PA, DR, B\rangle$, where $UR \subseteq \mathcal{U} \times \mathcal{R}$ is user-role membership relation, $DR \subseteq \mathcal{U} \times \mathcal{U} \times \mathcal{R} \times \{\text{“}g\text{”}, \text{“}t\text{”}\}$ is delegation relation, and B is a set of binary relations between users.

The representation of access control state in Definition 5.1.1 is similar to that in Definition 4.1.1, except that a delegation relation DR is included in the new definition. In the delegation relation DR, $(u_1, u_2, r, \text{“}g\text{”})$ indicates that u_1 has delegated the role r to u_2 via a *grant* operation, while $(u_1, u_2, r, \text{“}t\text{”})$ indicates that u_1 has delegated the role r to u_2 via a *transfer* operation. The difference between grant and transfer will be discussed later in this section.

Given a state γ, each user has a set of roles for which the user is authorized. A user is authorized for a role r if and only if he/she is a member of r or he/she received r from

another user through delegation. We formalize this by defining a function $authR : \mathcal{U} \times \Gamma \rightarrow 2^{\mathcal{R}}$, where Γ is the set of all states.

$$authR(u, \langle UR, PA, DR, B \rangle) = \{r \mid (u, r) \in UR \\ \vee \exists_{u'}((u', u, r, \text{“}g\text{”}) \in DR \vee (u', u, r, \text{“}t\text{”}) \in DR)\}$$

When a user u is authorized for the role r, he/she is authorized for the permissions assigned to r.

Next, we introduce the notations related to delegation. Assume that $Alice$ delegates the role `Accountant` to Bob. In such an operation, $Alice$, who is the granter of privilege, is called *delegator*; Bob, who is the receiver of privilege, is called *delegatee*; the role `Accountant` is the *delegated privilege*. We assume that each delegation operation has only one delegated privilege. If a user wants to delegate multiple privileges to the same receiver, he/she can perform multiple delegation operations.

A delegation operation is essentially an access control state transition operation, which takes one of the following three forms:

- $grant(u_1, u_2, r)$: user u_1 *grants* role r to user u_2. After the delegation operation, u_2 gains r and u_1 still keeps r.
- $trans(u_1, u_2, r)$: user u_1 *transfers* role r to user u_2. After the delegation operation, u_2 gains r and u_1 (temporarily) loses r.
- $revoke(u_1, u_2, r)$: user u_1 *revokes* the delegated privilege, role r, from u_2.

Note that a user can grant or transfer only the roles he/she is a member of to others. To simplify delegation relation, we assume that a delegatee cannot further delegate the delegated privilege to other users, and only the corresponding delegator can revoke the delegated privilege from the delegatee.

Since delegation is performed in a distributed manner, in the sense that everyone may perform delegation operations, it is undesirable to allow a user to delegate his/her roles in a completely unrestricted way. Delegation operations are thus subject to the control of authorization rules, which takes one of the following three forms:

- $can\_grant(cond, r)$: a user who satisfies condition $cond$ can grant r to other users, where $cond$ is an expression formed using roles, the binary operators $\wedge$ and $\vee$, the unary operator $\neg$, and parentheses.

- $can\_transfer(cond, r)$: a user who satisfies condition $cond$ can transfer r to other users.

- $can\_receive(cond, r)$: a user who satisfies condition $cond$ can receive r from other users.

  For example, the rule $can\_receive(\texttt{Clerk} \wedge \neg\texttt{Treasurer}, \texttt{Accountant})$ states that anyone who is a member of `Clerk` but not a member of `Treasurer` can receive the role `Accountant`.

Definition 5.1.2 (Administrative State) An *administrative state* consists of a set RL of authorization rules. Given RL, a delegation operation $grant(u_1, u_2, r)$ (or similarly, $trans(u_1, u_2, r)$) succeeds in the state $\langle UR, PA, DR, B\rangle$ if and only if

$$(u_1, r) \in UR \wedge can\_grant(c_1, r) \in RL \wedge (u_1 \text{ satisfies } c_1)$$
$$\wedge\, can\_receive(c_2, r) \in RL \wedge (u_2 \text{ satisfies } c_2)$$

Otherwise, the delegation operation fails.

To simplify management, we assume that if a user u_1 granted or transferred a role r to u_2 and has not revoked r from u_2 yet, then u_1 can neither grant nor transfer r to u_2 again. That is to say, at any moment, a user may receive a role from the same user at most once. But a user may receive the same role from different users.

We use $\gamma \rightarrow_{op}^{RL} \gamma'$ to denote the state transition from γ to γ' after applying the delegation operation op under administrative state RL. Let $\gamma = \langle UR, PA, DR, B\rangle$. The state transition rules are described as follows:

- $op = grant(u_1, u_2, r)$: If op fails, then $\gamma' = \gamma$. Otherwise, $\gamma' = \langle UR, PA, DR', B\rangle$, where $DR' = DR \cup \{(u_1, u_2, r, \text{“}g\text{”})\}$.

- If $op = trans(u_1, u_2, r)$: If op fails, then $\gamma' = \gamma$. Otherwise, $\gamma' = \langle UR', PA, DR', B\rangle$, where $UR' = UR/\{u_1, r\}$ and $DR' = DR \cup \{(u_1, u_2, r, \text{“}t\text{”})\}$.

- If $op = revoke(u_1, u_2, r)$: There are three cases. Let $\gamma' = \langle UR', PA, DR', B\rangle$.

  - If $(u_1, u_2, r, \text{“}g\text{”}) \in DR$, then $UR' = UR$ and $DR' = DR/\{(u_1, u_2, r, \text{“}g\text{”})\}$.
  - If $(u_1, u_2, r, \text{“}t\text{”}) \in DR$, then $UR' = UR \cup \{(u_1, r)\}$ and $DR' = DR/\{(u_1, u_2, r, \text{“}t\text{”})\}$.
  - Otherwise, $\gamma' = \gamma$. It indicates that u_2 did not receive r from u_1 in γ, and thus the revocation fails.

Note that PA and B are not affected by state transition rules.

With the above state transition rules, we may apply a sequence Q of delegation operations one by one to γ and acquire γ'. We say that γ' is *reachable* from γ under administrative state RL, which is denoted as $\gamma \rightsquigarrow_Q^{RL} \gamma'$.

An access control system with delegation support is defined in below.

Definition 5.1.3 An access control system is represented as a 3-tuple $\langle \gamma, W, RL\rangle$, where γ is the initial access control state, W is a set of workflows and RL is the administrative state.

We assume that in the initial state $\gamma = \langle UR, PA, DR, B\rangle$ of an access control system, we always have $DR = \emptyset$. That is to say, no delegation operations have been performed in the initial state.

5.1.1 Circumventing Security Policies Using Delegation

In this section, we consider how malicious users may collude to circumvent security policies in access control systems. We present two examples describing two scenarios, in which colluding users successfully complete those tasks that they would not be able to complete without the “help” of delegation. After each example, we summarize the characteristic of the attack in the scenario.

Example 7 In an institution, a sensitive task t must be completed by a *single* user who is a member of both roles r_1 and r_2. Task t is modeled as workflow $w_1 = \langle S, \preceq, SA, C \rangle$, where

$$S = \{s_1, s_2\}, s_1 \prec s_2$$
$$C = \{\langle = (s_1, s_2) \rangle\}$$

Permissions to perform s_1 and s_2 are assigned to r_1 and r_2, respectively. The constraint in C requires that the two steps must be performed by the same user, which enforces that an instance of w_1 can be completed only by a user who is a member of both r_1 and r_2.

$Alice$ and Bob are employees of the institution. $Alice$ is a member of r_1 but not r_2, while Bob is a member of r_2 but not r_1. Clearly, neither $Alice$ nor Bob is qualified to complete an instance of w_1. However, if $Alice$ delegates (either by grant or transfer) r_1 to Bob, then Bob is authorized to perform both s_1 and s_2 and he is thus able to complete an instance of w_1. In other words, if $Alice$ and Bob collude, they can complete a task which they should not be able to complete.

In Example 7, $Alice$ "lends" her role membership of r_1 to Bob to make him more "powerful" than before. The example demonstrates that, using delegation, a group of colluding users may create a "more powerful" user by aggregating role memberships of different individuals in the group. In that case, security policies that require a single user (rather than multiple users) with multiple role memberships to complete a task could be circumvented.

Example 8 In a company, the task of issuing checks is modeled as a workflow consisting of two steps s_{pre} and s_{app}, which stand for "check preparation" and "approval", respectively. In order to prevent fraudulent transactions, s_{pre} and s_{app} must be performed by two *different* members of the role `Treasurer` (or two `Treasurers` for short). The workflow can be represented as $w_2 = \langle S, \preceq, SA, C \rangle$, where

$$S = \{s_{pre}, s_{app}\}, s_{pre} \prec s_{app}$$
$$C = \{\langle \neq (s_{pre}, s_{app}) \rangle\}$$

Also, for the sake of resiliency, the company allows a `Treasurer` to transfer his/her role to a `Clerk` in case he/she is not able to work due to sickness or some

other reasons. In other words, $can\_transfer(\texttt{Treasurer}, \texttt{Treasurer}) \in RL$ and $can\_receive(\texttt{Clerk}, \texttt{Treasurer}) \in RL$.

$Alice$ and Bob are employees of the company and they decided to collude to issue checks for themselves. $Alice$ is a `Treasurer`, while Bob is a `Clerk` and is thus not qualified to perform any step in w_2. To achieve the goal, $Alice$ and Bob do the followings:

1. $Alice$ performs $trans(Alice, Bob, \texttt{Treasurer})$, which makes Bob a member of the role `Treasurer`.

2. Bob performs s_{pre} to prepare a check for $Alice$.

3. $Alice$ performs $revoke(Alice, Bob, \texttt{Treasurer})$ to revoke `Treasurer` from Bob and regains the role.

4. $Alice$ performs s_{app} to approve the check prepared by Bob.
   What the workflow system sees is that s_{pre} and s_{app} are performed by two different users. Thus, the constraint $\langle \neq (s_{pre}, s_{app}) \rangle$ is satisfied and the operation succeeds.

After all of the above being done, a check is issued and $Alice$ and Bob may share the money.

In Example 8, $Alice$'s role membership of `Treasurer` is used twice by two different users in the same workflow instance. This example demonstrates that colluding users can make "copies" of their access privileges using delegation to bypass security constraints that enforce separation of duty.

5.1.2 Formal Definition of Security

We have seen examples on how colluding users may circumvent security policies in access control systems with the help of delegation. It is clear that if an access control system allows colluding users to bypass security policies, then the system is insecure. But, how can we tell whether a security policy has been circumvented by delegation operations?

What should a "secure" system look like? We answer these fundamental questions by formally defining the notion of security with respect to delegation.

First of all, we present a general definition of security, which is independent of the concrete design of access control systems. Given an access control system, we define the predicate $can\_complete$, such that $can\_complete(t, U_1, U_2, \gamma)$ is "true" if and only if users in U_1 together can complete task t when the initial access control state is γ and only users in U_2 can perform delegation operations. The concrete definition of $can\_complete$ depends on how tasks are modeled and the concrete design of access control systems. We say that a group of users becomes more powerful (or gain power enhancement) when they eventually complete a task that they are not able to complete in the initial state (delegation is needed to change the state in this case). Intuitively, if an access control system is secure with respect to delegation, then a group of users cannot enhance the power of the group by performing delegation operations within the group. The following definition formally states such an intuition.

Definition 5.1.4 (Delegation Security) An access control system with initial access control state γ is *secure with respect to delegation* if and only if the following is true:

$$\forall_{t \in \mathcal{T}} \forall_{U \subseteq \mathcal{U}}\ can\_complete(t, U, U, \gamma) \Rightarrow can\_complete(t, U, \emptyset, \gamma)$$

where $\mathcal{T}$ is the set of all tasks and $\mathcal{U}$ is the set of all users in the system.

In the above definition, $can\_complete(t, U, U, \gamma)$ is "true" if and only if users in U together can complete t when the initial state is γ and delegation is available in such a way: the users may perform delegation operations to change the access control state, but no user outside of U is allowed to perform delegation operations. That is to say, users in U cannot get "help" from outsiders. In contrast, $can\_complete(t, U, \emptyset, \gamma)$ is "true" if and only if users in U together can complete t in state γ and no delegation operation is allowed. In general, Definition 5.1.4 essentially states that, in a secure access control system, if a set of users can complete a task without receiving any privilege from outsiders, then they must be able to compete the task without delegation at all. That is to say, delegation does not enable a set of users to enhance their own power by themselves.

The notion of security introduced in Definition 5.1.4 respects the definition of delegation. Delegation is defined as a mechanism that allows a user A to act on another user B's behalf by making B's access rights available to A. Let γ and γ' be the states before and after a delegation operation from B to A, respectively. The fact that A is working on B's behalf in γ' indicates that A should not be able to do more than A and B together (i.e. $\{A, B\}$) can do in γ. Furthermore, since B does not gain anything by delegating his/her privileges to A, $\{A, B\}$ in γ' cannot be more powerful than $\{A, B\}$ in γ. By generalizing such an argument to groups with arbitrary number of users, we acquire the notion of security in Definition 5.1.4.

We now illustrate the effect of delegation in a secure access control system by giving an example. Assume that $Alice$ grants (or transfers) a role r to Bob. Then, Bob may become more powerful by acquiring r. Furthermore, every group G such that $Bob \in G$ and $Alice \notin G$ may become more powerful as well, because one of its member (Bob) received a privilege from an outsider ($Alice$). However, every group G' such that $Alice, Bob \in G'$ should not gain power enhancement. Otherwise, G' enhances its own power after a delegation operation between its members and the access control system is insecure by Definition 5.1.4. In general, in a secure access control system, a group of users may gain power enhancement only if they receive privileges from outsiders.

Definition 5.1.4 is general and independent of concrete access control systems. In this chapter, tasks are modeled as workflows. We provide a more concrete definition of security for workflow authorization systems in below.

Definition 5.1.5 (Delegation Security for Workflow) An access control system $\langle \gamma, W, RL \rangle$ is *secure with respect to delegation* if and only if an adversary can never win the following one-person game.

Round 0:

The adversary selects a workflow $w \in W$ and a set U of users, such that U cannot complete w in γ without delegation. If such a combination of w and U does not exist,

then the adversary loses (in this case, the system is trivially secure as everyone is able to complete every task).

$PP \leftarrow \emptyset$ and $SS \leftarrow S$, where PP records past user-step assignments and SS records the remaining steps.

$i \leftarrow 1$ and $\gamma_0 \leftarrow \gamma$.

Round i:

1. The adversary designs a sequence Q_i of delegation operations such that every delegation operation in Q_i involves only users in U [1]. The adversary applies Q_i to γ_{i-1} and acquires a new state γ_i.
2. The adversary selects a step s from SS such that $\forall_{s' \in S}(s' \prec s \Rightarrow s' \notin SS)$. The adversary selects a user u from U as well.

   If u is not authorized for s in γ_i, then the adversary loses.

   Otherwise, $PP \leftarrow PP \cup \{(u, s)\}$ and $SS \leftarrow SS/\{s\}$.
3. If $SS = \emptyset$, then

   If no constraint in C is violated by PP, then

   The adversary wins;

   Otherwise, the adversary loses.

   Otherwise, $i \leftarrow i + 1$ and the game continues to the next round.

Note that in the above game, the effect of delegation operations is subject to RL. The adversary can perform a sequence of delegation operations to change the access control state at the beginning of each round. The game allows delegation operations between the execution of two steps (i.e. between two rounds) so that users can perform revocation to regain the roles that were transferred to other users in previous rounds. This gives the adversary more advantage than allowing the adversary to perform delegation operations

[1] We may allow users in U delegate privileges to outsiders. But this does not help the adversary to win the game.

only at the beginning of the game. In Example 8, delegation operations are performed between the execution of two steps.

The adversary winning the game indicates that there exit a group of users that can enhance themselves with the help of delegation. In that case, the access control system is vulnerable to collusion and is thus insecure with respect to delegation.

5.2 Enforcing the Security of Delegation

We have defined the formal notion of security with respect to delegation. A natural next step is to study mechanisms to enforce security. In this section, we study three enforcement approaches. In Section 5.2.1, we study static enforcement, in which security is ensured by careful design of administrative state. In Section 5.2.2, we discuss dynamic enforcement, where a verification procedure is performed by the end of the execution of each workflow instance to ensure that the participants have not enhanced their own power through delegation. In Section 5.2.3, we propose a third approach, the source-based enforcement mechanism, which employs a novel security policy evaluation method that is customized for delegation.

5.2.1 Static Enforcement

Given a set of workflows and an initial access control state, a straightforward approach to enforce security is to carefully design the administrative state RL so that no "dangerous" delegation operation would succeed. For instance, in Example 7, if RL does not allow members of r_2 to receive r_1 and vice versa, the collusion between $Alice$ and Bob could not succeed. Such an enforcement mechanism is called *static enforcement*, as the security of the system relies on (administrative) state configuration and can be verified in an off-line manner. An access control system that enforces security via a static enforcement mechanism is called a *statically secure system*. The following example illustrates the idea of static enforcement.

Example 9 The set W consists of a single workflow $w = \langle S, \preceq, SA, C \rangle$, where

$$S = \{s_1, s_2, s_3\}, s_1 \prec s_2 \prec s_3$$
$$C = \{\langle \neq (s_1, s_2) \rangle\}$$

Let γ be an access control state such that $UR = \{(Alice, r_1), (Bob, r_1), (Carl, r_2)\}$ and the permissions to perform s_1 and s_2 are assigned to r_1, while the permission to perform s_3 is assigned to r_2.

It is easy to see that $\{Alice, Bob, Carl\}$ is the only set of users in γ that can complete w. We can have $Alice$ performed s_1, Bob performed s_2 and $Carl$ performed s_3.

An access control system $\langle \gamma, W, RL \rangle$ is statically secure if and only if neither of the followings is true:

- $can\_grant(r_1, r_1) \in RL \wedge can\_receive(r_2, r_1) \in RL$
- $can\_transfer(r_1, r_1) \in RL \wedge can\_receive(r_2, r_1) \in RL$

To see this, on the one hand, if RL contains both $can\_grant(r_1, r_1)$ and $can\_receive(r_2, r_1)$, then $Alice$ (or Bob) can first complete s_1 and then grant r_1 to $Carl$, who can then complete s_2 and s_3. This violates the security requirement, because $\{Alice, Carl\}$ cannot complete w in γ without delegation. The system is vulnerable to the collusion between $Alice$ and $Carl$ (or Bob and $Carl$). Similar argument holds when RL contains both $can\_transfer(r_1, r_1)$ and $can\_receive(r_2, r_1)$.

On the other hand, if RL does not contain both rules, then $Carl$ cannot acquire r_1 from $Alice$ (or Bob). In that case, both $Alice$ and Bob must be involved to complete s_1 and s_2 so as to satisfy the constraint $\langle \neq (s_1, s_2) \rangle$. Hence, even if $Alice$ and $Carl$ (or Bob and $Carl$ collude), they cannot complete w. Also, $Alice$ and Bob together cannot complete w, because $Carl$ is the only member of r_2. Therefore, no size-2 sets of users may enhance their own power by delegation.

We would like to point out that it is fine that RL contains both $can\_grant(r_2, r_2)$ (or $can\_transfer(r_2, r_2)$) and $can\_receive(r_1, r_2)$. In that case, $Carl$ can grant r_2 to $Alice$

and/or Bob, so that $\{Alice, Bob\}$ can complete w. This does not violate the security requirement, because $Carl$ is involved through delegation and $\{Alice, Bob, Carl\}$ can complete w in γ.

The advantage of static enforcement is that, if we have already implemented an access control system with delegation support, we just need to modify the administrative state to enforce security. There is no need to change the existing implementation. However, static enforcement could make the administrative state more restrictive than necessary. For instance, assume that there are two workflows w_1 and w_2 in the system. $Alice$ and Bob are two users who are not supposed to complete w_1. But the system setting is such that if $Alice$ can successfully grant or transfer role r to Bob, then $Alice$ and Bob together can complete w_1. In order to prevent the potential collusion between $Alice$ and Bob, the administrative state must prevent $Alice$ from delegating r to Bob. But this is too restrictive as Bob may only intend to perform w_2 (instead of w_1) after receiving r, which could be allowed. But static enforcement mechanism does not take the actual usage of delegated privileges into account. Finally, the design of the administrative state is usually subject to administrative policies as well as practical considerations. It may be undesirable to dramatically alter the administrative state due to security concerns, for security should not significantly affect the usability of the system.

5.2.2 Dynamic Enforcement

Static enforcement is too restrictive as it does not take into account how delegatees use the delegated privileges. This motivates the proposal of dynamic enforcement for delegation security.

To begin with, we describe the high-level idea of dynamic enforcement. In dynamic enforcement, the initial state γ of the access control system is recorded. For every workflow instance X, the system maintains a list U_X of the participants for the instance. Every user who executed a step of X is added to U_X. When a user u requests to execute a step s, the system checks whether he/she needs to use a delegated privilege. If a delegated privilege r

should be used by u to perform s, then both u and the delegator of the privilege are added to U_X. Note that if u has received r from multiple delegators, u has to specify the delegator of r for the execution of s. At the end of the instance, the system checks whether the users in U_X can complete the workflow in γ without delegation. If they can, then the execution of X is confirmed. Otherwise, the system gives warning that users in U_X have enhanced their own power through delegation. The execution of X is rejected.

The problem of checking whether a set of users can complete a workflow in an access control state without delegation is called the *Workflow Satisfaction Problem (WSP)*.

Definition 5.2.1 (Workflow Satisfaction Problem) Given a set U of users, a workflow $w = \langle S, \preceq, SA, C \rangle$ and an access control state γ, the *Workflow Satisfaction Problem (WSP)* asks whether we can assign a user $u \in U$ to every step $s \in S$ such that u is authorized for s in γ and no constraint in C is violated by the overall assignments.

An instance of WSP is denoted as $\mathsf{wsp}(U, w, \gamma)$.

Detailed description of dynamic enforcement is given in Figure 5.1. Dynamic enforcement ensures that a workflow instance may be successfully completed only if the participants (including those users who perform a step and those delegators who contribute necessary privileges through delegation operations) can complete the same workflow instance in the initial state. Hence, the correctness of dynamic enforcement follows directly from Definition 5.1.4.

Dynamic enforcement monitors the usage of delegated privileges rather than placing restrictions on administrative states. It is thus less restrictive and more practical than static enforcement. However, dynamic enforcement introduces a performance overhead as the system needs to solve a WSP instance by the end of every workflow instance. According to Theorem 4.2.1, WSP is $\mathbf{NP}$-complete, which indicates that the runtime overhead of dynamic enforcement for each workflow instance could be exponential in the size of the workflow.

In real-world, the number of steps in a workflow is normally small. Hence, it is possible that the performance of dynamic enforcement is acceptable in practice. Also, dynamic

Let γ be the initial state of the access control system. For every workflow instance, the system does the followings. Let X be an instance of workflow w.

- When X is created: $U_X \leftarrow \emptyset$
- When a step s is performed by a user u: Let p_s be the permission to perform s and $\gamma' = \langle UR, PA, DR, B \rangle$ be the current state.
  - If there exists a role r such that $((u, r) \in UR \wedge (p_s, r) \in PA)$, then $U_X \leftarrow U_X \cup \{u\}$.

    This indicates that u can use his/her own privilege to perform the step.
  - Otherwise, u specifies a user u' such that $((u', u, r) \in DR \wedge (p_s, r) \in PA)$. $U_X \leftarrow U_X \cup \{u, u'\}$.

    This indicates that u is using a delegated privilege r received from u' to perform the step. When the choice of u' and r is unique, the system may do the selection itself rather than asking the user to specify the choice.
- After X is finished: The system solves $\mathsf{wsp}(U_X, w, \gamma)$. If the answer to $\mathsf{wsp}(U_X, w, \gamma)$ is "yes", then the result of X is confirmed; otherwise, the result of X is voided and necessary roll-back is performed.

Figure 5.1. Description of dynamic enforcement

enforcement does not require changing existing implementation of workflow modules. All we need to do is to add a module to the system to perform recording and the closing verification procedure for workflow instances.

5.2.3 Source-Based Enforcement

We have discussed two mechanisms to enforce delegation security in access control systems. Even though both approaches have the advantage of allowing the reuse of existing workflow implementation, they have major drawbacks: static enforcement is too restrictive and dynamic enforcement may introduce large performance overhead. A natural question is, if we are willing to redo the workflow module, can we have a better mechanism to enforce delegation security?

In this section, we propose the source-based enforcement mechanism, which employs a novel method to evaluate constraints in workflow systems. We describe the idea of source-based enforcement mechanism by presenting a design of a secure workflow system. Our workflow system is secure with respect to Definition 5.1.5 and introduces almost no performance overhead.

The high-level idea of source-based enforcement is that, when a user $Alice$ requests to perform a step s of a workflow instance, he/she must specify the privilege to be used and the source of the privilege. For instance, assume that $Alice$ requests to perform a step s with role r. If $Alice$ is a member of r, then $Alice$ may specify herself as the source of r. If $Alice$ received r from others, then $Alice$ may pick a delegator of r and specify the delegator as the source. Note that, even if $Alice$ is a member of r herself, she may still specify another user as the source of r as long as she has received r from that user.

Given the privilege r and its source u_o specified by $Alice$, the system checks the constraints on s as if it is u_o rather than $Alice$ who is performing s. For example, assume that workflow w consists of two steps s_1 and s_2, both of which can be performed by members of role `Accountant`. There is a constraint in w, which states that the users who perform s_1 and s_2 must not have conflicts of interests. Assume that $Alice$ has executed s_1 using her own

membership of `Accountant`. Now, $Carl$ tries to use the delegated privilege `Accountant` received from Bob to perform s_2. Instead of checking conflict of interests between $Carl$ and $Alice$ as what traditional workflow systems do, our system checks conflict of interests between Bob and $Alice$. The intuition is that, since $Carl$ is using a delegated privilege from Bob, he is working on Bob's behalf. Hence, Bob and $Alice$ must not have conflicts of interests. By evaluating constraints in this way, we can ensure that the system is secure with respect to delegation.

Sometimes, in addition to sources of privileges, we want to take the actual performers into account while evaluating constraints. To achieve this, our system supports two types of constraints. Type-1 constraint only ensures that the sources of privileges satisfy the constraint; Type-2 constraint is more restrictive: if either the actual performer or the source violates the constraint, then the constraint is violated. For instance, if the constraint in the example in the previous paragraph is a Type-2 constraint, then $Alice$ must not have conflict of interests with either Bob (source) or $Carl$ (actual performer).

Next, we describe the design of a secure workflow system, which employs the source-based enforcement mechanism.

System Description: The system adopts the representations of access control state and the state transition rules introduced in Section 5.1. The only major change in this system is the way workflow constraints are evaluated.

A workflow is represented as $\langle S, \preceq, SA, C\rangle$, where S is a set of steps, $\prec \subseteq S \times S$ defines a partial order among steps in S, and C is a set of constraints. $s_1 \prec s_2$ indicates that s_1 must be performed before s_2.

A *constraint* takes the form of $\langle \rho(s_1, s_2, i)\rangle$ where s_1 and s_2 are two steps, ρ is a binary relation between users and $i = 1$ or 2. When $i = 1$, the constraint is of *Type-1*, while when $i = 2$, the constraint is of *Type-2*.

Let $w = \langle S, \preceq, SA, C\rangle$. $\gamma = \langle UR, PA, DR, B\rangle$ is the current access control state. When a user u requests to perform a step s of an instance X of w, u presents a pair $\langle u_o, r\rangle$,

where u_o is a user identity and r is a role. u_o is called the *source* of r. The pair $\langle u_o, r\rangle$ is valid if and only if one of the followings is true:

- $u = u_o \wedge (u, r) \in UR$. In other words, u is using his own role membership to perform s.
- $u \neq u_o \wedge ((u_o, u, r, \text{“}g\text{”}) \in DR \vee (u_o, u, r, \text{“}t\text{”}) \in DR)$. That is to say, u_o has granted or transferred r to u and u requests to perform s on u_o's behalf.

With the pair $\langle u_o, r\rangle$, u can successfully execute s if and only if both of the followings hold:

1. u is authorized to perform s with role r. That is, $(p_s, r) \in PA$, where p_s is the permission to perform s.
2. No constraint is violated. That is, for every constraint c on s:
   - Case $c = \langle \rho(s, s', 1)\rangle$: $(u_o, u'_o) \in \rho$
     where u'_o is the source of the privilege used to perform s'
   - Case $c = \langle \rho(s', s, 1)\rangle$: $(u'_o, u_o) \in \rho$
     where u'_o is the source of the privilege used to perform s'
   - Case $c = \langle \rho(s, s', 2)\rangle$:

     $$(u, u') \in \rho \wedge (u_o, u') \in \rho \wedge (u, u'_o) \in \rho \wedge (u_o, u'_o) \in \rho$$

     where u' is the user who actually performed s' and u'_o is the source of the privilege used to perform s'.
   - Case $c = \langle \rho(s', s, 2)\rangle$:

     $$(u', u) \in \rho \wedge (u', u_o) \in \rho \wedge (u'_o, u) \in \rho \wedge (u'_o, u_o) \in \rho$$

     where u' is the user who actually performed s' and u'_o is the source of the privilege used to perform s'.

Note that in the first two cases, c is a Type-1 constraint and only the sources must satisfy the constraint. In the latter two cases, c is a Type-2 constraint, and both the sources and the actual performers are taken into account.

After a step is executed, the system records the identities of both the actual performer and the source of privilege for future reference.

The following example illustrates how the system works.

Example 10 In a bank, task t is modeled as a workflow $w = \langle S, \preceq, SA, C \rangle$, where

$$S = \{s_1, s_2\}, s_1 \prec s_2$$
$$C = \{\langle \neq (s_1, s_2, 1) \rangle\}$$

The permissions to perform s_1 and s_2 are assigned to r_1 and r_2, respectively. $Alice$ is a member of r_1 and Bob is a member of r_2.

$Alice$ becomes too busy to work on t and would like to balance the workload with Bob by delegating r_1 to Bob. Let X be an instance of w. Bob performs s_1 in X by presenting $\langle Alice, r_1 \rangle$ to the system. The system records that Bob is the actual performer of s_1 in X and $Alice$ is the source of privilege. Next, Bob requests to perform s_2 in X by presenting $\langle Bob, r_2 \rangle$, which indicates that himself is the source of r_2. The system found that the constraint $\langle \neq (s_1, s_2, 1) \rangle$ needs to be checked. Since the constraint is of Type-1, the system only considers the sources of privilege for s_1 and s_2, which are $Alice$ and Bob respectively. Because $Alice \neq Bob$, the constraint is satisfied, and Bob completes X. Note that this does not violate the notion of security, because $Alice$ is involved in X by allowing Bob to work on her behalf, and $Alice$ and Bob together can complete w before the delegation operation.

Now, assume that the constraint in C is of Type-2 (i.e. $\langle \neq (s_1, s_2, 2) \rangle$). In this case, Bob cannot complete w. When $\langle \neq (s_1, s_2, 2) \rangle$ is checked, the system takes both the actual performers and the sources into account. When the system compares the actual performer of s_1 with the source of privilege (or the actual performer) of s_2, it has $Bob = Bob$, which indicates that $Bob \neq Bob$ does not hold. Hence, the constraint is violated and Bob is rejected from performing s_2.

It is clear that Type-2 constraints provide stronger security than Type-1 constraints. People may wonder why we support the seemingly less secure Type-1 constraints in our system. First of all, as we will prove later in this section, Type-1 constraints are sufficient to enforce the notion of security defined in Definition 5.1.5. Secondly, in certain situations, we may gain flexibility by using Type-1 constraints. For instance, a workflow may have a constraint c stating that s_1 and s_2 must be performed by the same user. Assume that $Alice$ has performed s_1 in an instance X of the workflow but she has to leave before performing s_2. If c is a Type-1 constraint (i.e. $c = \langle = (s_1, s_2, 1)\rangle$), then $Alice$ may delegate her privilege r to another user Bob who may complete s_2 in X by presenting the pair $\langle Alice, r\rangle$ to the system; but if c is a Type-2 constraint, then s_2 of X cannot be completed until $Alice$ comes back. In situations where it is more beneficial to complete the task, we should declare c as Type-1. In particular, in order for an access control state to be resilient to user absence, all binding of duty constraints (i.e. constraints that use $=$) must be Type-1 constraints. In contrast, in situations where security is given high priority and we would rather have the task uncompleted than allow another user to involve, we should declare c as Type-2. The choice between Type-1 and Type-2 constraints can be viewed as a flexibility-security trade-off. Our system provides the options and leave the decisions to security policy designers.

Next, we prove that our workflow system is secure with respect to delegation. The overall idea of the proof is that, for every workflow instance that is completed, we modify its user-step assignment by replacing the actual performer of each step with the corresponding source of privilege. Since our constraint evaluation procedure always takes sources into account, the modified user-step assignment must be valid for the workflow in the initial state of the system. This implies that the set of sources can complete the workflow in the initial state. Recall that in the initial state of the system, $DR = \emptyset$. Also, recall that we assume that a delegatee cannot further delegate the delegated privilege to others. At the end of this section, we will discuss extending our system to allow further delegation of privileges.

Theorem 5.2.1 The workflow system employing source-based enforcement mechanism is secure with respect to delegation.

Proof We show that any workflow system, all of whose constraints are of Type-1, is secure with respect to delegation. This is sufficient to prove that workflow systems using only Type-2 constraints or both types of constraints are secure as well, because Type-2 constraints are more restrictive than Type-1 constraints.

Given a workflow system in which all constraints are of Type-1, assume for the purpose of contradiction that the adversary can win the game in Definition 5.1.5. Let γ be the initial state of the system, $w = \langle S, \preceq, SA, C \rangle$ be the workflow and U be a set of users selected by the adversary. By Definition 5.1.5, U cannot complete w in γ without delegation (i.e. $\mathsf{wsp}(U, w, \gamma)$ is false). The user-step assignment for instance X of w has been stored in PP. We define a predicate $Source(s, X)$, which returns the source of the privilege used to perform step s in workflow instance X.

We construct another user-step assignment PP' for w such that for every step s, $(u, s) \in PP \Rightarrow (Source(s, X), s) \in PP'$. Next, we show that PP' is a valid user-step assignment for w in γ.

First of all, for every step s in w, we show that $Source(s, X)$ is authorized to perform s in γ. Let $Source(s, X) = u_o$. $(u, s) \in PP$ implies that u is authorized to perform s in an access control γ', which is reachable from γ. Let $\gamma = \langle UR, PA, DR, B \rangle$ and $\gamma' = \langle UR', PA, DR', B \rangle$. If $u_o = u$, then the statement is trivially true, because $UR \supseteq UR'$. Otherwise, there exists a role r, such that $(p_s, r) \in PA$ and $(u_o, u, r) \in DR'$, where p_s is the permission to perform s. $(u_o, u, r) \in DR'$ implies that $(u_o, r) \in UR$, because a user can only delegate the roles which he/she is a member of. In general, u_o is authorized to perform s in γ.

Second, for every constraint $c \in C$, we show that PP' satisfies c. Let $c = \langle \rho(s_1, s_2, 1) \rangle$. Since c is Type-1, the system takes into account the sources of privileges for s_1 and s_2 whenever c is checked. Hence, the fact that PP does not violate c indicates that $(Source(s_1, X), Source(s_2, X)) \in \rho$. Therefore, PP' does not violate c.

In general, in PP', all steps in w are assigned to authorized users and no constraint in C is violated. That is to say, users in U can complete w in γ, which contradicts the assumption that $\mathsf{wsp}(U, w, \gamma)$ is false. Therefore, the adversary could not have won the game and the workflow system is secure. ■

The design of our secure workflow system is based on a delegation model where a delegatee cannot further delegate the privileges he/she received to other users. We can easily extend the design to apply to situations where further delegation is supported. In those cases, the system maintains chains of delegation operations. For instance, assume that u_1 grants role r to u_2, who further grants r to u_3. When u_3 uses r to perform a step in a workflow instance, u_1 is considered to be the source of r, and should be taken into account in constraint checking so as to ensure security. We may design different types of constraints that take the actual performer (i.e. u_3) and even intermediate users in the delegation chain (e.g. u_2) into consideration. We believe the ideas introduced in the source-based enforcement mechanism will also be useful in the design of other types of access control systems that support delegation.

5.3 Enforcing Resiliency Using Delegation

We have studied the security of delegation in workflow authorization systems. In this section, we discuss how to enforce resiliency using delegation.

When Alice is about to be absent, she may select a suitable person based on the current situations and delegate her privileges to that person so that the latter may perform Alice's tasks on her behalf. This may allow the system be resilient to Alice's absence. Consider the following example.

Example 11 Task t is modeled as workflow $w_1 = \langle S, \preceq, SA, C \rangle$, where

$$S = \{s_1, s_2\}, s_1 \prec s_2$$
$$C = \{\langle \rho(s_1, s_2, 2) \rangle\}$$

Permissions to perform s_1 and s_2 are assigned to r_1 and r_2, respectively. Bob and Carl are members of r_1, while Alice is a member of r_2. Also, we have $(Bob, Alice), (Bob, Dave), (Carl, Alice), (Carl, Elaine) \in \rho$.

Since Alice is the only user in the system who is authorized to perform s_2, she would like to delegate r_2 to another user before she leaves. Alice may choose who she should delegate r_2 to based on the situation right before she leaves. If Bob performed step s_1 of the workflow instance, Alice may delegate r_2 to Dave so that Dave can finish the workflow without violating the constraint in C, as we have $(Bob, Dave) \in \rho$ (recall that for type-2 constraint, both the sources of privilege and actual performers must have the corresponding relation). In contrast, if it was Carl who performed s_1, Alice should delegate r_2 to Elaine instead, because $(Carl, Elaine) \in \rho$.

Sometimes unexpected events may happen and Alice may be absent without specifying how to delegate her privileges. A natural solution to handle unexpected absence of users is to have default delegation plans for users so that when a user is absent, the delegated operations in the corresponding default delegation plan is executed to delegate the absent user's privileges to other users. Certain commercial access control systems employ default delegation plans determined by software developers. For example, using IBM Tivoli Identity Manager (ITIM) [28], if a user does not perform a task assigned to her after a certain period of time, her supervisor will be asked to perform the task for her. This can be viewed as the absent user's privilege to perform the task is automatically delegated to her supervisor. In ITIM, one cannot change the default delegation plan. This is problematic in cases where one's supervisor is not the right person to perform a task on one's behalf. In this section, we consider a more flexible approach by allowing every user to specify her own default delegation plan.

A default delegation plan of a user, say Alice, is a set of delegation operations which will be performed automatically when Alice is absent. A delegation operation in the plan specifies which one of Alice's privileges will be delegated to whom. The delegation operations in the plan must not violate delegation administrative rules in the system so as to be valid. Note that Alice may specify more than one delegation operations for the same

role, and she may choose one of the following three options for the multiple delegation operations regarding role r: (1) All: all the delegation operations for r will be executed when Alice is absent; in other words, r will be delegated to multiple users. (2) First: the delegation operations for r are ordered, and when Alice is absent, the first operation whose delegatee is available will be executed. (3) Any: Alice lets the system to choose which delegation operations for r to be executed; the system will make selections based on realtime situations. For example, if Bob is very busy when Alice is absent, the system may execute the operation that delegates r to Carl rather than the one that delegates r to Bob.

In this dissertation, we assume that every user designs their own default delegation plans independently. This is the simplest approach and is easy to carry out in practice. A more complex approach is to ask users to collaborate in their delegation planning. For example, assume that Alice and Bob are both members of role r and they would like to delegate r to either Carl or Dave. If Bob delegates role r to Carl in his plan, it may be better for Alice to delegate r to Dave rather than to Carl so that even if Alice, Bob, and Carl are all absent, there is still a member of r (i.e. Dave) in the system. Even though the collaborative approach can potentially achieve higher level of resiliency, it may be difficult to carry out in practice due to a number of reasons. First, it takes more effort for a user to setup her plan as she has to coordinate with others. Second, one may have to give up her favorite choices to achieve better overall results, and some users may be reluctant to do so. Third, if a user change her plan later, then multiple related plans of other users' may need to be changed as well, which leads to large overhead and inconvenience. Due to such practical difficulties, we leave the collaborative approach as interesting future work.

A natural question arises is how does a user design a "good" default delegation plan for herself. On the one hand, to be compliant with the principle of least privilege, one may not want to delegate too many privileges to others. On the other hand, one would like to have all the necessary delegation operations in her plan to make the system be resilient to the absence of herself. In the following, we provide a guideline on designing an effective default delegation plan.

Assume that Alice would like to design a default delegation plan so as to let the system be resilient to her absence. Given an access control state $\langle UR, PA, DR, B\rangle$, for every workflow W and every step s in W that Alice is authorized to performed, we ask the following questions in order. Let r be the role that is authorized to perform s.

1. Is Alice the only one who is authorized to perform s? In other words, is Alice the only member of role r?

   Yes: A delegation operation for role r is recommended to be added to Alice's default delegation plan. Otherwise, when Alice is absent, no one is authorized to perform s and thus W cannot be completed.

   No: Go to the next question.

2. Is there any constraint on s?

   No: A delegation operation for r may not be necessary. Since there is no constraint on s, another member of r can perform s even if Alice is absent.

   Yes: Go to the next question.

3. For every constraint $c = \langle \rho(s', s, i)\rangle$ in W where $i = 1$ or $i = 2$, let $U_{s'}$ and U_s be the set of users who are authorized to perform s' and s, respectively. Is there a user $u \in U'$ such that Alice is the only one in U_s such that $(u, Alice) \in \rho$?

   Yes: A delegation operation for role r is recommended. Otherwise, if u performs s' and Alice becomes absent before performing s, then no one is able to perform s without violating c. Recall that a type-1 constraint only requires the sources of privilege satisfy the relation. So, if $i = 1$, Alice may delegate r to any user (without violating administrative rules) and the user who receives r from Alice is able to perform s while satisfying c. In contrast,if $i = 2$, then Alice must delegate r to a user u_1 such that $(u, u_1) \in \rho$ so that u_1 may perform s when u performs s'.

Note that $c = \langle = (s', s, 1)\rangle$ is a special case where we may have $u = Alice$. And if $c = \langle = (s', s, 2)\rangle$, then delegation cannot help complete W if Alice becomes absent after performing s' but not s.

No: No recommendation on whether to setup a delegation operation for r. When this branch is reached, there must exist a constraint in the form of $\langle \rho(s, \exists X, i)\rangle$ or in the form of $\langle \rho(s', \exists X, i)\rangle$ where $s \in X$. Alice may further examine the privileges and relations of users authorized to perform steps in X and s' so as to determine whether a delegation operation for r is necessary to make the system be resilient to her absence. But the number of users that need to be checked may be large. Thus, we do not provide a suggestion in this case and leave the decision to the user.

Furthermore, when Alice is considering adding a delegation operation for role r to her default delegation plan, she may check if existing operations in her plan for role r suffice to meet her goal. If so, she may avoid setting up another delegation operation for r.

We have provided a guideline for users to design their default delegation plans. In practice, users may have other considerations when setting up their delegation plans. An interesting question is to check whether the system meets resiliency requirements after default delegation plans have been set up for all users. We study such a problem in the rest of this section. We assume that an absent user does not perform any delegation operation when the user becomes absent, except that her default delegation plans are automatically executed. Note that such a problem is computationally at least as difficult as the corresponding resiliency checking problem without delegation, as the latter is a special case of the former when the default delegation plan for every user is empty.

Static resiliency In static resiliency, the set of absent users is known before the execution of a workflow. To check that if the system is statically resilient to the absence of a certain set U' of users with regards to a workflow W, we may remove users in U' and execute their default delegation plans to change the access control state. We then check whether

the remaining users can complete W in the new access control state, which is an instance of the WSP problem.

Decremental resiliency In decremental resiliency, users may become absent during the execution of a workflow instance and absent users will not come back. An algorithm, which checks whether a system is decrementally resilient to the absence of k users with respect to workflow W, is given in Figure 5.2. Intuitively, we enumerate all possible scenarios of the absence of k users; when a user u is absent, we execute her default delegation plan and go on to check if the system is resilient to the absence of $k-1$ users for the remaining steps of W in the new access control state.

Dynamic resiliency In dynamic resiliency, up to k users may be absent when a step in the workflow is performed and absent users may return after the step is finished. An algorithm, which checks that if a system is dynamically resilient to the absence of k users with respect to workflow W, is given in Figure 5.3. Intuitively, we try every possible way to remove a set U' of k users before the execution of a step, execute the default delegation plans for users in U', assign a step to an available user, and then repeat such a process for the remaining steps.

Note that the above algorithms employ a brute-force strategy. In practice, the probability that multiple users become absent simultaneously or within a short period of time is small. Hence, we are mostly interested in the case where $k = 1$. Also, the number of steps in W is normally small. Hence, the performance of the above algorithms should be acceptable in real-world scenarios.

The function *check-decremental* checks if the access control state γ is decrementally resilient to the absence of k users with regards to workflow W, where U is the set of available users in γ, S is the set of incomplete steps in W, and P is the current partial plan. $Auth(s, U, \gamma)$ is the set of users in U that are authorized to perform step s in state γ.

Function *check-decremental*(γ, U, k, W, S, P)
  If $S = \emptyset$ Then
    Return "yes";
  If exists $s \in S$ such that $Auth(s, U, \gamma) = \emptyset$ Then
    Return "no";
  If $k > 0$ Then
    For every $u_i \in U$ Do
      Execute the default delegation plan of u_i to change γ to γ'
      If (*check-decremental*(γ', $U - \{u_i\}$, k, W, S, P) == "no") Then
        Return "no";
    EndFor;
  EndIf;
  For every $s \in S$ that is ready to be executed based on P
    For every $u \in Auth(s, U, \gamma)$ Do
      $P' = P \cup \{(s, u)\}$;
      If (*check-decremental*(γ', U, $k - 1$, W, $S - \{s\}$, P') == "yes") Then
        Return "yes";
    EndFor;
  EndFor;
  Return "no";
End

Figure 5.2. Algorithm for checking decremental resiliency with default delegation plans

The function *check-dynamic* checks if the access control state γ is dynamically resilient to the absence of k users with regards to workflow W, where U is the set of available users in γ and S is the set of incomplete steps in W. $Auth(s, U, \gamma)$ is the set of users in U that are authorized to perform step s in state γ.

Function *check-dynamic*(γ, U, k, W, S, P)
  If $S = \emptyset$ Then
    Return "yes";
  For every size-k subset U' of U Do
    Execute the default delegation plans of users in U' to change γ to γ';
    If (*make-assignment*($\gamma', U - U', k, W, S, P$) == "no") Then
      Return "no";
  EndFor;
  Return "yes";
End

Function *make-assignment*(γ, U, k, W, S, P)
  For every $s \in S$ that is ready to be executed based on P
    For every $u \in Auth(s, U, \gamma)$ Do
      $P' = P \cup \{(s, u)\}$;
      If (*check-dynamic*(γ, U, k, W, $S - \{s\}$, P') == "yes") Then
        Return "yes";
    EndFor;
  EndFor;
End

Figure 5.3. Algorithm for checking dynamic resiliency with default delegation plans

6 RELATED WORK

6.1 Access Control Policy Specification

The concept of SoD has long existed in the physical world, sometimes under the name "the two-man rule", for example, in the banking industry and the military. To our knowledge, in the information security literature the notion of SoD first appeared in Saltzer and Schroeder [3] under the name "separation of privilege." Clark and Wilson's commercial security policy for integrity [2] identified SoD along with well-formed transactions as two major mechanisms of fraud and error control. Nash and Poland [29] explained the difference between dynamic and static enforcement of SoD policies. In the former, a user may perform any step in a sensitive task provided that the user does not also perform another step on that task. In the latter, users are constrained a-priori from performing certain steps.

Sandhu [30,31] presented Transaction Control Expressions, a history-based mechanism for dynamically enforcing SoD policies. A transaction control expression associates each step in the transaction with a role. By default, the requirement is such that each step must be performed by a different user. One can also specify that two steps must be performed by the same user. In Transaction Control Expressions, user qualification requirements are associated with individual steps in a transaction, rather than a transaction as a whole.

Li et al [20] studied both direct and indirect enforcement of static separation of duty (SSoD) policies. They showed that directly enforcing SSoD policies is intractable (NP-complete). They also discussed using static mutually exclusive roles (SMER) constraints to indirectly enforce SSoD policies. They defined what it means for a set of SMER constraints to precisely enforce an SSoD policy, characterize the policies for which such constraints exist, and show how they are generated. In Section 2.2, we study the enforcement of policies specified in our algebra, which include SoD policies as a sub-class; however, our computational results (those on SSC) are on direct static enforcement only.

There exists a wealth of literature on constraints in the context of RBAC [32–39]. They either proposed and classified new kinds of constraints [35, 38] or proposed new languages for specifying sophisticated constraints [32–34, 37, 39]. Most of these constraints are motivated by SoD and are variants of role mutual exclusion constraints, which may declare two roles to be mutually exclusive so that no user can be a member of both roles.

McLean [19] introduced a framework that includes various mandatory access control models. Security models are instances of the framework; and they differ in which users are allowed to change the security levels. These models form a boolean algebra. McLean also looked at the issue of N-person policies, where a policy may allow multiple subjects acting together to perform some action. McLean adopted the monotonicity requirement in such N-person policies. McLean [19] does not discuss how to specify N-person policies, and the examples in the paper list explicitly the usersets that are allowed access. Our algebra, on the other hand, is about how to define policies that require multiple users with qualification requirements.

Abadi et al. [40] developed a calculus for access control in distributed systems. The calculus allows compound principals to be formed from basic ones using two operations $\wedge$ (and) and $|$ (quoting). Some principals are groups, when a principal u is a member of a group g, then u speaks for g. One can express multi-user policies in this calculus. An access control policy is specified as an access control list (ACL), where each entry is an expression in the calculus. The $\wedge$ corresponds to $\odot$ in our algebra. That is, if an ACL entry contains $g_1 \wedge g_2$, then a single user that is a member of both g_1 and g_2 is allowed access, as are two users such that one is a member of g_1 and the other is a member of g_2. The $\sqcup$ operator in our algebra can be partially supported in the calculus by having multiple ACL entries, which has the effect of supporting logical OR, but only at the top level. The other operators $\neg$, $^+$, $\sqcap$ and $\otimes$ cannot be expressed in the calculus.

Several algebras have been proposed for combining security policies. These include the work by Bonatti et al. [41, 42], Wijesekera and Jajodia [43], Pincus and Wing [44]. These algebras are designed for purpose that are different from ours; therefore, they are quite different from our algebra. Each element in their algebra is a policy that specifies

what subjects are allowed to access which resources, whereas each element in our algebra maps to a user.

The two operators $\odot$ and $\otimes$ in our algebra are taken from the RT family of role-based trust-management languages designed by Li et al. [45]. In [45], the notion of manifold roles was introduced, which are roles that have usersets, rather than individual users, as their members. The two operators $\otimes$ and $\odot$ are used to define manifold roles. Our work differs in that we propose to combine these two operators together with four other operators $\sqcup$, $\sqcap$, $\neg$, and $+$ (which are not in RT) in an algebra for specifying high-level security policies. In addition, we also study the algebraic properties of these operators, the satisfaction problems, and the term satisfiability problem related to the algebra.

Readers who are familiar with description logic (DL) may find similarities between the algebra and DL. However, there is a fundamental difference between the two: a term in DL describes *a set of individuals*, while a term in the algebra describes *a set of sets of individuals*. A concept in DL defines a set of individuals, which corresponds to a role in the algebra; a "role" in DL defines a binary relation between individuals. DL supports operators $\neg, \sqcap$ and $\sqcup$, which stand for complement of concepts, intersection of concepts and union of concepts, respectively. If we interpret a unit term in our algebra as a set of individuals[1], then unit terms may be viewed as a strict subset of terms in DL. But in general case, there is no operator in DL that corresponds to operators $^+, \odot$ and $\otimes$ in our algebra. Hence, computational complexity problems studied in this chapter are not directly related to those in DL.

6.2 Access Control in Workflow Systems

Bertino et al. [5] introduced a language to express workflow authorization constraints as clauses in a logic programming language. The language supports a number of predefined relations for constraint specification. Bertino et al. [5] also proposed searching algorithms to assign users to complete a workflow. This work does not support user-defined binary

[1] A unit term in our algebra describes a set of singletons.

relations, nor does it formally study computational complexity of the workflow satisfiability problem. Tan et al. [7] studied the consistency of authorization constraints in workflow systems. The model in [7] supports six predefined binary relations: $\{=, \neq, <, \leq, >, \geq\}$, but not user-defined relations. Atluri and Huang [4] proposed a workflow authorization model that focuses on temporal authorization. This model does not support constraints about users performing different steps in a task. In [8], Warner and Atluri considered authorization constraints that span multiple instances of a workflow. Their model supports predefined relations with emphasis on inter-instance constraints. Inter-instance problems in workflow systems is an interesting research area. The model in [8] does not support user-defined relations. Finally, Kang et al. [46] investigated access control mechanisms for inter-organizational workflow. Their workflow model authorizes steps to roles and supports dynamic constraints. However, they do not explicitly point out how constraints are specified and what kinds of constraints are supported besides separation of duty. Their paper mainly focuses on infrastructure design and implementation.

The workflow authorization model proposed by Crampton [6] is probably the one that is most closely related to R^{2}BAC. The model in [6] supports user-defined binary relations; however, it does not support quantifiers in constraints, so that constraints of the form $\langle \rho(\exists X, s) \rangle$ cannot be expressed in that model. Crampton [6] also studied the workflow satisfiability problem and presented a polynomial time algorithm for their model. However, the algorithm is incorrect.[2] Each constraint in [6] relates two steps in an workflow. The algorithm (Figure 2 in [6]) tries to gradually reduce the set of users that can be applied to each step. One first calculates the set of authorized users for each individual step, and then for each constraint that involves steps s_1 and s_2, one remove from the sets for steps s_1 and s_2 those users that cannot be paired with a user satisfying the constraint. If no set can be reduced further and no set is empty, the algorithm declares that a workflow is satisfiable. The problem with this algorithm is that, while it ensures that each individual constraint can be satisfied, it does not ensure the combination of them can. For a counter example, consider a workflow with 4 steps and 3 users, where every user is authorized to perform every step.

[2]We have verified the bug with the author of [6].

The constraints are such that no two steps can be performed by the same user. Obviously, a valid execution assignment would not exist. However, the algorithm would return true. As we have pointed out in Theorem 4.2.6, the workflow satisfiability problem is **NP**-hard in general for any workflow model that supports user-inequality constraints. Since the model in [6] supports such type of constraints, a polynomial time algorithm for the satisfiability problem in their model could not exist.

None of the work mentioned above has given the computational complexity results of the Workflow Satisfiability Problem, whereas we give a clear characterization using parameterized complexity. Also, the resiliency problem in workflows has not been studied before in the literature.

6.3 Delegation in Access Control

Delegation has received considerable attention from the research community. In [9,10], Barka and Sandhu proposed a framework for role-based delegation models (RBDM), which identifies a number of characteristics related to delegation. Example characteristics are (1) *monotonicity*: grant is a monotonic delegation operation, while transfer is non-monotonic; (2) *totality*: whether one can delegate only a portion of the permissions of a role rather than the entire role; (3) *levels of delegation*: the number of times delegatees may further delegate the received privileges. Many characteristics identified by RBDM are used in delegation models proposed later.

There exist a wealth of delegation models in literature [11–17]. L. Zhang et al. [13] presented a role-based delegation model called RDM2000. Their model supports the specification of delegation authorization rules to impose restrictions on which roles can be delegated to whom. They proposed a language to specify and enforce rules regarding how users delegate their roles and how delegated roles can be revoked. Furthermore, they have designed and implemented a web-based application as a prototype of their delegation framework.

In RDM2000, the unit of delegation is a role. X. Zhang et al. [12] proposed a role-based delegation model called PBDM, which supports both role and permission level delegation.

Their model controls delegation operations through the notion of delegatable roles such that only permissions assigned to these roles can be delegated to others. Another model that supports permission-level delegation was proposed by Wainer and Kumar in [14]. Their model also supports constraints (or rules) that determine whether a user can receive a certain right through delegation.

A delegation operation can either be a grant or a transfer operation. Most exiting work either focuses on grant or does not explicitly distinguish the two types of operations. In [17], Crampton and Khambhammettu proposed a delegation model that supports both grant and transfer. Furthermore, they proposed to use administrative scope in role hierarchy to determine delegation authorization rules.

Atluri and Warner [15] studied how to support delegation in workflow systems. They extended the notion of delegation to allow conditional delegation, where conditions can be based on time, workload and task attributes. One may specify rules to determine under what condition a delegation operation should be performed. For example, $Alice$ may specify a rule to delegate a task t to Bob when she has five or more tasks assigned. Such rules may result in cycles (e.g. u_1 delegates t to u_2, who further delegates t to u_3, who delegates t back to u_1). To address this issue, they studied the delegation consistency problem which determines whether it is possible to satisfy all rules in the system.

All the above work mainly focus on the modeling and management of delegation, while our dissertation focuses on the security impact of delegation on access control systems. None of the above work proposes a formal notion of security regarding delegation or studies mechanisms to enforce security in access control systems with delegation support.

In [47], Shaad observed that delegation and revocation features of a system may be used to circumvent separation of duty properties. He gave an example to illustrate an attack conducted by a single user. In his example, there is a separation of duty policy which requires that no single user may first access an object o using privilege $auth_1$ and then access o again with privilege $auth_2$. The system he designed enforces such a policy by allowing a user to access o only if the user does not have both $auth_1$ and $auth_2$ at the time of access. Let $Alice$ be a malicious user having both $auth_1$ and $auth_2$. $Alice$ first transfers

$auth_2$ to another user Bob so as to temporarily lose $auth_2$. Next, she accesses o with $auth_1$ and then revokes $auth_2$ from Bob to regain the privilege. Finally, $Alice$ transfers $auth_1$ to Bob and then accesses o again using $auth_2$. In this case, the separation of duty policy is circumvented. This example differs from our examples in Section 5.1.1 in a couple of ways:

1. The attack in [47] is conducted by a single user ($Alice$), as the delegatee (Bob) is not actively involved. In contrast, our examples are on multi-user collusion, where all principles are actively involved in the attack.

2. The attack in [47] relies on a specific way in which separation of duty is implemented. In particular, it is assumed that the system does not maintain any historical record. But this is not the case in most of the existing workflow authorization systems [5–8], as these systems keep track of which users have performed which steps so as to enforce constraints. In contrast, our examples apply to workflow authorization systems in existing literature.

In general, the example in [47] has a very different nature from our examples in Section 5.1.1. Shaad's paper [47] is about an access control framework and the interaction between delegation and security policies is not the main focus of the paper. Problems such as collusion and enforcement mechanisms for security, which are studied in our dissertation, are not discussed in [47].

6.4 Other Related Work on Policy Analysis

In [48], Li and Tripunitara proposed the notion of security analysis in RBAC. They formally defined the notions of RBAC states, state transition rules and a family of security analysis problems. An example security analysis problem is whether a certain set of users can gain a certain set of permissions in a state that is reachable from the initial state. However, in workflow systems, possessing the set of necessary permissions is not sufficient for the users to complete a workflow, as there may be constraints on the relations of users

performing different steps. For instance, if there is a constraint requiring Step 1 and Step 2 be performed by different users, then even if $Alice$ has permissions to perform both steps, she is not able to complete both of them by herself. Li and Tripunitara did not consider workflows and constraints in [48]. Hence, the security regarding delegation in workflow systems is beyond the security analysis problems proposed in [48].

In [49], Stoller et al also applied parameterized complexity theory to computational problems on access control policy analysis. They studied policies in Administrative RBAC (ARBAC), while we focused on workflow authorization systems, especially in the R^2BAC model in Chapter 4. Policies in ARBAC are role-based, while security constraints in R^2BAC are based on binary relations between users. The parameterized complexity results in [49] and those in Chapter 4 of this dissertation cannot be easily reduced to each other.

7 SUMMARY

In this dissertation, we have proposed an algebra as a novel policy-specification language, resiliency policy as a new family of access control policies, and R^2BAC as a new access control model for workflows. We have also studied a number of fundamental policy-analysis problems, such as the workflow satisfiability problem and the security of delegation. The contributions of this dissertations are summarized as follows.

- We have proposed a novel algebra that enables the specification of high-level security policies that combine qualification requirements with quantity requirements. Our algebra contains six operators and is expressive enough to specify many natural high-level security policies. We have studied the algebraic properties of the algebra, as well as several computational problems related to the algebra.

- We have formally defined resiliency policies, which require an access control system to be resilient to the absence of users. We have studied computational problems on checking whether an access control state satisfies a resiliency policy. We have also studied the consistency between resiliency policies and separation of duty policies.

- We have proposed the role-and-relation-based access control (R^2BAC) model for workflow authorization systems. In R^2BAC, in addition to a user's role memberships, the user's relationships with other users help determine whether the user is allowed to perform a certain step in a workflow.

- We have studied fundamental problems in workflow authorization systems, such as determining whether a set of users can complete a workflow and checking whether a workflow is resilient to the absence of users. In particular, we have applied tools from parameterized complexity theory to better understand the complexities of the workflow satisfiability problem.

- We have studied the impact of delegation on the security of workflow authorization systems. We have formally defined the notion of security with respect to delegation and proposed mechanisms to enforce delegation security in workflow authorization systems. We have also discussed how to use delegation to meet resiliency requirements.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2001.

[2] David D. Clark and David R. Wilson. A comparision of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 184–194. IEEE Computer Society Press, May 1987.

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

[4] V. Atluri and W. Huang. An authorization model for workflows. In *Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS)*, pages 44–64, 1996.

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

[6] Jason Crampton. A reference monitor for workflow systems with constrained task execution. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 38–47, Stockholm, Sweden, June 2005.

[7] K. Tan, J. Crampton, and C. Gunter. The consistency of task-based authorization constraints in workflow systems. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 155–169, 2004.

[8] Janice Warner and Vijayalakshmi Atluri. Inter-instance authorization constraints for secure workflow management. In *Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 190–199, 2006.

[9] E. Barka and R. Sandhu. Framework for role-based delegation models. In *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC)*, page 168, Washington, DC, USA, 2000. IEEE Computer Society.

[10] E. Barka and R. Sandhu. A role-based delegation model and some extensions. In *Proceedings of the 23rd National Information Systems Security Conference*, 2000.

[11] SangYeob Na and SuhHyun Cheon. Role delegation in role-based access control. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, pages 39–44, New York, NY, USA, 2000. ACM Press.

[12] Xinwen Zhang, Sejong Oh, and Ravi Sandhu. Pbdm: a flexible delegation model in RBAC. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 149–157, New York, NY, USA, 2003. ACM Press.

[13] Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu. A rule-based framework for role-based delegation and revocation. *ACM Transactions on Information and System Security*, 6(3):404–441, 2003.

[14] Jacques Wainer and Akhil Kumar. A fine-grained, controllable, user-to-user delegation method in rbac. In *Proceedings of the 10th ACM symposium on Access Control Models and Technologies (SACMAT)*, pages 59–66, New York, NY, USA, 2005. ACM Press.

[15] Vijayalakshmi Atluri and Janice Warner. Supporting conditional delegation in secure workflow management systems. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 49–58, New York, NY, USA, 2005. ACM.

[16] James B. D. Joshi and Elisa 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 (SACMAT)*, pages 81–90, New York, NY, USA, 2006. ACM Press.

[17] J. Crampton and H. Khambhammettu. Delegation in role-based access control. In *Proceedings of 11th European Symposium on Research in Computer Security (ESORICS)*, 2006.

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

[19] John McLean. The algebra of security. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 2–7, April 1988.

[20] Ninghui Li, Mahesh V. Tripunitara, and Ziad Bizri. On mutually exclusive roles and separation-of-duty. *ACM Transactions on Information and Systems Security (TISSEC)*, 10(2):5, 2007.

[21] Adam Barth. Managing digital rights using linear logic. In *In Proceedings of the 21st Symposium on Logic In Computer Science (LICS)*, pages 127–136. IEEE Computer Society Press, 2006.

[22] Butler W. Lampson. Protection. In *Proceedings of the 5th Princeton Conference on Information Sciences and Systems*, 1971. Reprinted in ACM Operating Systems Review, 8(1):18-24, Jan 1974.

[23] G. Scott Graham and Peter J. Denning. Protection — principles and practice. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 40, pages 417–429. AFIPS Press, May 16–18 1972.

[24] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley Longman, 1994.

[25] Michael R. Garey and David J. Johnson. *Computers And Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[26] Ninghui Li, Ziad Bizri, and Mahesh V. Tripunitara. On mutually-exclusive roles and separation of duty. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 42–51. ACM Press, October 2004.

[27] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer, 1999.

[28] Axel Buecker, Jaime Cordoba Palacios, Brian Davis, Todd Hastings, and Ian Yip. *Identity Management Design Guide with IBM Tivoli Identity Manager*. IBM.

[29] Michael J. Nash and Keith R. Poland. Some conundrums concerning separation of duty. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 201–209, May 1990.

[30] Ravi Sandhu. Separation of duties in computerized information systems. In *Proceedings of the IFIP WG11.3 Workshop on Database Security*, September 1990.

[31] Ravi S. Sandhu. Transaction control expressions for separation of duties. In *Proceedings of the 4th Annual Computer Security Applications Conference (ACSAC)*, December 1988.

[32] Gail-Joon Ahn and Ravi S. Sandhu. The RSL99 language for role-based separation of duty constraints. In *Proceedings of the 4th Workshop on Role-Based Access Control*, pages 43–54, 1999.

[33] Gail-Joon Ahn and Ravi S. Sandhu. Role-based authorization constraints specification. *ACM Transactions on Information and System Security*, 3(4):207–226, November 2000.

[34] Jason Crampton. Specifying and enforcing constraints in role-based access control. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 43–50, Como, Italy, June 2003.

[35] Virgil D. Gligor, Serban I. Gavrila, and David F. Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 172–183, May 1998.

[36] Trent Jaeger. On the increasing importance of constraints. In *Proceedings ACM Workshop on Role-Based Access Control (RBAC)*, pages 33–42, 1999.

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

[38] Richard T. Simon and Mary Ellen Zurko. Separation of duty in role-based environments. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 183–194. IEEE Computer Society Press, June 1997.

[39] Jonathon Tidswell and Trent Jaeger. An access control model for simplifying constraint expression. In *Proceedings of ACM Conference on Computer and Communications Security*, pages 154–163, 2000.

[40] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, October 1993.

[41] Piero Bonatti, Sabrina de Capitani di Vimercati, and Pierangela Samarati. A modular approach to composing access control policies. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 164–173, November 2000.

[42] Piero Bonatti, Sabrina de Capitani di Vimercati, and Pierangela Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 5(1):1–35, February 2002.

[43] Duminda Wijesekera and Sushil Jajodia. A propositional policy algebra for access control. *ACM Transactions on Information and Systems Security (TISSEC)*, 6(2):286–325, May 2003.

[44] Jon Pincus and Jeannette M. Wing. Towards an algebra for security policies (extended abstract). In *Proceedings of ICATPN 2005*, number 3536 in LNCS, pages 17–25. Springer, 2005.

[45] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[46] Myong H. Kang, Joon S. Park, and Judith N. Froscher. Access control mechanisms for inter-organizational workflow. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 66–74, New York, NY, USA, 2001. ACM.

[47] Andreas Schaad. A framework for organisational control principles. In *PhD Thesis, University of York*, 2003.

[48] Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. *ACM Transactions on Information and Systems Security (TISSEC)*, 9(4):391–420, November 2006.

[49] Scott D. Stoller, Ping Yang, C R. Ramakrishnan, and Mikhail I. Gofman. Efficient policy analysis for administrative role based access control. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, pages 445–455, New York, NY, USA, 2007. ACM.

[50] Christos H. Papadimitrou and Kenneth Steiglitz. *Combinatorial Optimization*. Prentice Hall, 1982.

APPENDICES

Appendix A Proofs in Chapter 2

A.1 Proofs of Theorems in Section 2.1

Proof of Lemma 2.1.1

Given a node N in T, let $\Phi(N)$ be the sub-term of ϕ represented by the sub-tree rooted at N. In the following, we prove by induction that for every node N, if $L_T(N) \neq \emptyset$, then $L_T(N)$ satisfies $\Phi(N)$.

Base case: When N is a leaf node, by Definition 2.1.4, $L_T(N)$ is either $\emptyset$ or it satisfies $\Phi(N)$. Since $L_T(N) \neq \emptyset$, $L_T(N)$ satisfies $\Phi(N)$.

Inductive case: Assume that the statement holds for both children N_1 and N_2 of N and $L_T(N) \neq \emptyset$.

- When N represents $\sqcap$: According to Definition 2.1.4, $L_T(N) = L_T(N_1) = L_T(N_2)$. Since $L_T(N) \neq \emptyset$, $L_T(N_1)$ and $L_T(N_2)$ are non-empty. By inductive assumption, $L_T(N_1)$ and $L_T(N_2)$ satisfy $\Phi(N_1)$ and $\Phi(N_2)$, respectively. Since $\Phi(N) = \Phi(N_1) \sqcap \Phi(N_2)$, by Definition 2.1.3, $L_T(N)$ satisfies $\Phi(N)$.

- When N represents $\sqcup$: According to Definition 2.1.4, $L_T(N) = L_T(N_1)$ or $L_T(N) = L_T(N_2)$. Without loss of generality, assume that $L_T(N) = L_T(N_1)$. Since $L_T(N) \neq \emptyset$, $L_T(N_1)$ is non-empty. By inductive assumption, $L_T(N_1)$ satisfies $\Phi(N_1)$. Since $\Phi(N) = \Phi(N_1) \sqcup \Phi(N_2)$, by Definition 2.1.3, $L_T(N)$ satisfies $\Phi(N)$.

- When N represents $\odot$: According to Definition 2.1.4, $L_T(N) = L_T(N_1) \cup L_T(N_2)$, and since $L_T(N) \neq \emptyset$, $L_T(N_1)$ and $L_T(N_2)$ are also non-empty. By inductive assumption, $L_T(N_1)$ and $L_T(N_2)$ satisfy $\Phi(N_1)$ and $\Phi(N_2)$, respectively. Since $\Phi(N) = \Phi(N_1) \sqcap \Phi(N_2)$, by Definition 2.1.3, $L_T(N)$ satisfies $\Phi(N)$.

- When N represents $\otimes$: This is very similar to the above case.

Proof of Theorem 2.1.2

Clearly, if there exists a satisfaction tree of ϕ with root labeled X, then X satisfies ϕ. Now we show the other direction. If a userset X satisfies ϕ under $\langle U, UR\rangle$, we construct a satisfaction tree for ϕ. First of all, we construct the syntax tree T of ϕ and label its root with X. We then recursively label other nodes in T in a top-down manner. Let N be an inner node labeled with a non-empty userset. We label the children N_1 and N_2 of N in the following manner.

- When N represents $\sqcap$: We label N_1 and N_2 with $L_T(N)$. This satisfies the rules specified in Definition 2.1.4. Since $L_T(N)$ satisfies $\Phi(N)$ and $\Phi(N) = \Phi(N_1) \sqcap \Phi(N_2)$, we have $L_T(N_1) = L_T(N)$ satisfies $\Phi(N_1)$ and $L_T(N_2) = L_T(N)$ satisfies $\Phi(N_2)$.
- When N represents $\sqcup$: Since $L_T(N)$ satisfies $\Phi(N)$ and $\Phi(N) = \Phi(N_1) \sqcup \Phi(N_2)$, either $L_T(N)$ satisfies $\Phi(N_1)$ or $L_T(N)$ satisfies $\Phi(N_2)$. Without loss of generality, assume that $L_T(N)$ satisfies $\Phi(N_1)$. We label N_1 with $L_T(N)$ and N_2 with $\emptyset$. We also label all the nodes in the sub-tree rooted at N_2 with $\emptyset$. This satisfies the rules specified in Definition 2.1.4.
- When N represents $\odot$: Since $L_T(N)$ satisfies $\Phi(N)$ and $\Phi(N) = \Phi(N_1) \odot \Phi(N_2)$, according to Definition 2.1.3, we have non-empty sets X_1 and X_2 such that $L_T(N) = X_1 \cup X_2$ and X_1 satisfies $\Phi(N_1)$ and X_2 satisfies $\Phi(N_2)$. We label N_1 with X_1 and N_2 with X_2. Since $L_T(N) = X_1 \cup X_2$, the labeling satisfies the rules specified in Definition 2.1.4.
- When N represents $\otimes$: we label it in ways similar to the above case.

According to the above, when X satisfies ϕ, we can construct a satisfaction tree whose root is labeled with X.

Proof of Theorem 2.1.3 on Algebraic Properties

1. The operators $\sqcup, \sqcap, \otimes, \odot$ are commutative and associative.

   This is straightforward from Definition 2.1.3.

2. The operator $\sqcup$ distributes over $\sqcap$.

   If a userset X satisfies $(\phi_1 \sqcup (\phi_2 \sqcap \phi_3))$, then either X satisfies ϕ_1, or X satisfies both ϕ_2 and ϕ_3. It follows that X satisfies $((\phi_1 \sqcup \phi_2) \sqcap (\phi_1 \sqcup \phi_3))$.

   If X satisfies $((\phi_1 \sqcup \phi_2) \sqcap (\phi_1 \sqcup \phi_3))$, then X satisfies $(\phi_1 \sqcup \phi_2)$ and $(\phi_1 \sqcup \phi_3)$. There are only two cases: (1) X satisfies ϕ_1; and (2) X satisfies both ϕ_2 and ϕ_3. In either case, X satisfies $(\phi_1 \sqcup (\phi_2 \sqcap \phi_3))$.

   The operator $\sqcap$ distributes over $\sqcup$.

   If X satisfies $(\phi_1 \sqcap (\phi_2 \sqcup \phi_3))$, then X satisfies both ϕ_1 and $(\phi_2 \sqcup \phi_3)$, which means X satisfies either ϕ_2 or ϕ_3. It follows that X satisfies $((\phi_1 \sqcap \phi_2) \sqcup (\phi_1 \sqcap \phi_3))$.

   If X satisfies $((\phi_1 \sqcap \phi_2) \sqcup (\phi_1 \sqcap \phi_3))$, then either (1) X satisfies $(\phi_1 \sqcap \phi_2)$ or (2) X satisfies $(\phi_1 \sqcap \phi_3)$. In both cases, X satisfies ϕ_1; furthermore, X satisfies either ϕ_2 or ϕ_3. It follows that X satisfies $(\phi_1 \sqcap (\phi_2 \sqcup \phi_3))$.

3. The operator $\odot$ distributes over $\sqcup$.

   If X satisfies $(\phi_1 \odot (\phi_2 \sqcup \phi_3))$, then there exist X_1 and X_2 such that $X_1 \cup X_2 = X$, X_1 satisfies ϕ_1, and X_2 satisfies $(\phi_2 \sqcup \phi_3)$. By Definition 2.1.3, X_2 satisfies ϕ_2 or ϕ_3. In the former case, X satisfies $(\phi_1 \odot \phi_2)$, which implies that X satisfies $((\phi_1 \odot \phi_2) \sqcup (\phi_1 \odot \phi_3))$, as desired. The argument is analogous if X_2 satisfies ϕ_3 but not ϕ_2.

   If X satisfies $((\phi_1 \odot \phi_2) \sqcup (\phi_1 \odot \phi_3))$, then either X satisfies $(\phi_1 \odot \phi_2)$ or X satisfies $(\phi_1 \odot \phi_3)$. Without loss of generality, assume that X satisfies $(\phi_1 \odot \phi_2)$, then there exist X_1, X_2 such that $X_1 \cup X_2 = X$, X_1 satisfies ϕ_1 and X_2 satisfies ϕ_2. Therefore, X_2 satisfies $(\phi_2 \sqcup \phi_3)$, and consequently, X satisfies $(\phi_1 \odot (\phi_2 \sqcup \phi_3))$ as desired.

4. The operator $\otimes$ distributes over $\sqcup$.

   If X satisfies $(\phi_1 \otimes (\phi_2 \sqcup \phi_3))$, X can be partitioned into two disjoint sets X_1 and X_2 such that X_1 satisfies ϕ_1 and X_2 satisfies ϕ_2 or ϕ_3. In this case, by definition, X satisfies $(\phi_1 \otimes \phi_2)$ or $(\phi_1 \otimes \phi_3)$, which means X satisfies $((\phi_1 \otimes \phi_2) \sqcup (\phi_1 \otimes \phi_3))$.

   For the other direction, if X satisfies $((\phi_1 \otimes \phi_2) \sqcup (\phi_1 \otimes \phi_3))$, it satisfies either $(\phi_1 \otimes \phi_2)$ or $(\phi_1 \otimes \phi_3)$. Without loss of generality, assume that X satisfies $(\phi_1 \otimes \phi_2)$. Then, X can be partitioned into two disjoint sets X_1 and X_2 such that X_1 satisfies ϕ_1 and X_2

satisfies ϕ_2. By definition, X_2 satisfies $(\phi_2 \sqcup \phi_3)$. Therefore, X satisfies $(\phi_1 \otimes (\phi_2 \sqcup \phi_3))$.

5. No other ordered pair of operators have the distributive property.

   We show a counterexample for each case. In the following, $U_r = \{u | (u, r) \in UR\}$.

   (a) The operator $\odot$ does not distribute over $\sqcap$.

   If X satisfies $(\phi_1 \odot (\phi_2 \sqcap \phi_3))$, then X also satisfies $((\phi_1 \odot \phi_2) \sqcap (\phi_1 \odot \phi_3))$.

   However, the other direction of implication does not hold. Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_1\}$, and $U_{r_3} = \{u_2\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \odot r_2) \sqcap (r_1 \odot r_3))$, but does not satisfy $(r_1 \odot (r_2 \sqcap r_3))$.

   (b) The operator $\sqcap$ does not distribute over $\odot$. Neither direction holds.

   Counterexample: Let $U_{r_1} = U_{r_3} = \{u_1\}$ and $U_{r_2} = U_{r_4} = \{u_2\}$, let $\phi_1 = (r_1 \odot r_2)$, then $\{u_1, u_2\}$ satisfies $(\phi_1 \sqcap (r_3 \odot r_4))$, but does not satisfy $((\phi_1 \sqcap r_3) \odot (\phi_1 \sqcap r_4))$.

   Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_1\}$, and $U_{r_3} = \{u_2\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \sqcap r_2) \odot (r_1 \sqcap r_3))$, but does not satisfy $(r_1 \sqcap (r_2 \odot r_3))$.

   (c) The operator $\sqcup$ does not distribute over $\odot$.

   If X satisfies $(\phi_1 \sqcup (\phi_2 \odot \phi_3))$, then X satisfies $((\phi_1 \sqcup \phi_2) \odot (\phi_1 \sqcup \phi_3))$.

   However, the other direction of implication does not hold. Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_1\}$ and $U_{r_3} = \{u_1\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \sqcup r_2) \odot (r_1 \sqcup r_3))$, but does not satisfy $(r_1 \sqcup (r_2 \odot r_3))$.

   (d) The operator $\sqcup$ does not distribute over $\otimes$. Neither direction holds.

   Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_1\}$ and $U_{r_3} = \{u_1\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \sqcup r_2) \otimes (r_1 \sqcup r_3))$, but does not satisfy $(r_1 \sqcup (r_2 \otimes r_3))$.

   Counterexample: Let $U_{r_1} = U_{r_2} = U_{r_3} = \{u_1\}$, then $\{u_1\}$ satisfies $(r_1 \sqcup (r_2 \otimes r_3))$, but does not satisfy $((r_1 \sqcup r_2) \otimes (r_1 \sqcup r_3))$.

   (e) The operator $\otimes$ does not distribute over $\sqcap$.

   If X satisfies $(\phi_1 \otimes (\phi_2 \sqcap \phi_3))$, then X satisfies $((\phi_1 \otimes \phi_2) \sqcap (\phi_1 \otimes \phi_3))$.

However, the other direction of implication does not hold. Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_1\}$ and $U_{r_3} = \{u_2\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \otimes r_2) \sqcap (r_1 \otimes r_3))$, but does not satisfy $(r_1 \otimes (r_2 \sqcap r_3))$.

(f) The operator $\sqcap$ does not distribute over $\otimes$. Neither direction holds.

Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_1\}$ and $U_{r_3} = \{u_2\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \sqcap r_2) \otimes (r_1 \sqcap r_3))$, but does not satisfy $(r_1 \otimes (r_2 \sqcap r_3))$.

Counterexample: Let $U_{r_1} = U_{r_3} = \{u_1\}$ and $U_{r_2} = U_{r_4} = \{u_2\}$, and let $\phi_1 = (r_1 \odot r_2)$, then $\{u_1, u_2\}$ satisfies $(\phi_1 \sqcap (r_3 \otimes r_4))$, but does not satisfy $((\phi_1 \sqcap r_3) \otimes (\phi_1 \sqcap r_4))$.

(g) The operator $\odot$ does not distribute over $\otimes$. Neither direction holds.

Counterexample: Let $U_{r_1} = \{u_1, u_4\}$, $U_{r_2} = \{u_2\}$ and $U_{r_3} = \{u_3\}$, then $\{u_1, u_2, u_3, u_4\}$ satisfies $((r_1 \odot r_2) \otimes (r_1 \odot r_3))$, but does not satisfies $(r_1 \odot (r_2 \otimes r_3))$.

Counterexample: Let $U_{r_1} = \{u_1\}$, $U_{r_2} = \{u_1\}$ and $U_{r_3} = \{u_2\}$, then $\{u_1, u_2\}$ satisfies $(r_1 \odot (r_2 \otimes r_3))$, but does not satisfy $((r_1 \odot r_2) \otimes (r_1 \odot r_3))$.

(h) The operator $\otimes$ does not distribute over $\odot$.

If X satisfies $(\phi_1 \otimes (\phi_2 \odot \phi_3))$, then X satisfies $((\phi_1 \otimes \phi_2) \odot (\phi_1 \otimes \phi_3))$.

However, the other direction of implication does not hold. Counterexample: Let $U_{r_1} = \{u_1, u_2\}$, $U_{r_2} = \{u_2\}$ and $U_{r_3} = \{u_1\}$, then $\{u_1, u_2\}$ satisfies $((r_1 \otimes r_2) \odot (r_1 \otimes r_3))$, but does not satisfy $(r_1 \otimes (r_2 \odot r_3))$.

6. $(\phi_1 \sqcap \phi_2)^+ \equiv (\phi_1^+ \sqcap \phi_2^+)$.

If a userset X satisfies $(\phi_1 \sqcap \phi_2)^+$, then for every $u \in X$, $\{u\}$ satisfies $(\phi_1 \sqcap \phi_2)$ and thus satisfies ϕ_1 and ϕ_2. Hence, X satisfies ϕ_1^+ and ϕ_2^+, which means that X satisfies $(\phi_1^+ \sqcap \phi_2^+)$.

If X satisfies $(\phi_1^+ \sqcap \phi_2^+)$, then X satisfies both ϕ_1^+ and ϕ_2^+. For every $u \in X$, $\{u\}$ satisfies both ϕ_1 and ϕ_2. Hence, X satisfies $(\phi_1 \sqcap \phi_2)^+$.

7. DeMorgan's Law: $\neg(\phi_1 \sqcap \phi_2) \equiv (\neg\phi_1 \sqcup \neg\phi_2)$, $\neg(\phi_1 \sqcup \phi_2) \equiv (\neg\phi_1 \sqcap \neg\phi_2)$

The proof is straightforward by definition of $\neg, \sqcap$ and $\sqcup$.

A.2 Proofs of Theorems in Section 2.3

In the following proofs, $(\mathsf{op}_k \phi)$ denotes k copies of ϕ connected together by operator op and $(\mathsf{op}_{i=1}^{n} r_i)$ denotes $(r_1\ \mathsf{op} \ldots \mathsf{op}\ r_n)$. Given $R = \{r_1, \ldots, r_m\}$, $(\mathsf{op} R)$ denotes $(r_1\ \mathsf{op} \ldots \mathsf{op}\ r_m)$.

A.2.1 Proof of Lemma 2.3.1, Lemma 2.3.2, and Theorem 2.3.4

Proof of Lemma 2.3.1

To prove that TSAT over terms built using only roles, $\neg$, $\sqcap$, and $\sqcup$ is NP-hard, we reduce the $\mathbf{NP}$-complete SAT problem to it. Given a propositional logic formula e, let $\{v_1, \ldots, v_n\}$ be the set of propositional variables that appear in e. Construct a term ϕ by substituting every occurrence of v_i $(i \in [1, n])$ in e with the atomic term r_i, every occurrence of $\neg v_i$ $(i \in [1, n])$ with $\neg r_i$, and replacing logical `AND` with $\sqcap$ and logical `OR` with $\sqcup$. The result is a unit term. By Definition 2.1.3, a term without $\odot, \otimes$ and $^+$ can be satisfied by singletons only. If ϕ is satisfiable, then there exists a configuration $\langle U, UR \rangle$ and a user u such that $\{u\}$ satisfies ϕ. We can construct a truth assignment T in which v_i is `TRUE` if and only if $(u, r_i) \in UR$. It is clear that e evaluates to `TRUE` under T. Similarly, if there exists a truth assignment T such that e evaluates to `TRUE` under T, we can construct UR in which u is a member of r_i if and only if v_i is `TRUE` in T. In that case, $\{u\}$ satisfies ϕ under $\langle U, UR \rangle$. Therefore, e is satisfiable if and only if ϕ is satisfiable.

Proof of Lemma 2.3.2

To prove that TSAT over terms built using only explicit sets of users, $\sqcap$, $\sqcup$, and $\odot$ is $\mathbf{NP}$-hard, we reduce the $\mathbf{NP}$-complete SET COVERING problem to it. In the SET COVERING problem, we are given a finite set $U = \{u_1, \ldots, u_n\}$, a family $F = \{U_1, \ldots, U_m\}$ of subsets of U, and an integer k no larger than m, and we ask whether there is a sub-family $F' \subseteq F$ of sets whose union is U and $|F'| \leq k$.

We view each element in U as a user. For every $j \in [1, m]$, we construct a term $\phi_j = \bigodot\{\{u_i\} \mid u_i \in U_j\}$; that is, $\phi_j = \{u_{j_1}\} \odot \{u_{j_2}\} \odot \cdots \odot \{u_{j_x}\}$, where $U_j =$

$\{u_{j_1}, u_{j_2}, \ldots, u_{j_x}\}$. It is clear that ϕ_j can only be satisfied by U_j. Finally, we construct a term $\phi = ((\bigodot_k(\bigsqcup_{i=1}^m \phi_i)) \sqcap (\bigodot_{i=1}^n \{u_i\}))$. Since $(\bigodot_{i=1}^n \{u_i\})$ can be satisfied only by U, U is the only userset that may satisfy ϕ.

We now demonstrate that ϕ is satisfiable if and only if there are no more than k sets in family F whose union is U. On the one hand, if ϕ is satisfiable, then it must be satisfied by U. In this case, U satisfies $(\bigodot_k(\bigsqcup_{i=1}^m \phi_i))$, which means that there exist k sets $U'_1, \ldots, U'_k$ such that $\bigcup_{i=1}^k U'_i = U$ and each U'_i satisfies $(\bigsqcup_{i=1}^m \phi_i)$. Since ϕ_i can be satisfied only by $U_i \in F$, we have $U'_j \in F$ for every $j \in [1, k]$. The answer to the SET COVERING problem is thus "yes". On the other hand, without loss of generality, assume that $\bigcup_{i=1}^k U_i = U$. We have, for every $i \in [1, k]$, U_i satisfies ϕ_i and thus satisfies $(\bigsqcup_{i=1}^m \phi_i)$. Therefore, U satisfies $(\bigodot_k(\bigsqcup_{i=1}^m \phi_i))$. Since U also satisfies $(\bigodot_{i=1}^n \{u_i\})$, U satisfies $((\bigodot_k(\bigsqcup_{i=1}^m \phi_i)) \sqcap (\bigodot_{i=1}^n \{u_i\}))$.

Proof of Lemma 2.3.3

First, assume that a userset X satisfies ϕ under $\langle U, UR\rangle$. According to Theorem 2.1.2, there exists a satisfaction tree T of ϕ under $\langle U, UR\rangle$ and $L_T(N_r) = X$. Now, we show that if $|X| > |\phi|$, then there must exist $X' \subseteq X$ such that X' satisfies ϕ under $\langle U, UR\rangle$ and $|X'| \leq |\phi|$. In the following, we construct a satisfaction tree T' of ϕ based on T.

Initially, $X' = \emptyset$. For every leaf node N_i of T, if $L_T(N_i) \neq \emptyset$, then we arbitrarily select $u \in L_T(N_i)$ and add u to X'. Since the number of leaves in T is no larger than $|\phi|$, we have $|X'| \leq |\phi|$. Also, $X' \subseteq X$ because $L_T(N_i) \subseteq L_T(N_r) = X$ according to Definition 2.1.4. Next, for every node N in T, we relabel N with $L_{T'}(N)$ such that $L_{T'}(N) = L_T(N) \cap X'$. When the relabeling is done, we acquire a new tree T'. In particular, the root of T' is labeled with $X \cap X' = X'$. Now, we show that T' is a satisfaction tree by proving that it satisfies the conditions in Definition 2.1.4. Given a node N in T, we denote $\Phi(N)$ as the sub-term of ϕ that is represented by the sub-tree rooted at N. When $L_T(N) = \emptyset$, $L_{T'}(N) = \emptyset$. In the following, we only discuss the cases when $L_T(N) \neq \emptyset$.

- When N is a leaf node: If $\Phi(N)$ is a unit term, then $L_T(N)$ must be a singleton and the only user in $L_T(N)$ must have been added to X'. Thus, we have $L_{T'}(N) = L_T(N)$

which satisfies $\Phi(N)$. Otherwise, $\Phi(N)$ is in the form of ϕ_1^+. $L_T(N)$ satisfying ϕ_1^+ indicates that every user in $L_T(N)$ satisfies ϕ_1. Since at least one user in $L_T(N)$ has been added to X', $L_{T'}(N) = L_T(N) \cap X'$ is a non-empty subset of $L_T(N)$. Therefore, $L_{T'}(N)$ satisfies ϕ_1^+.

- When N represents $\sqcap$: Because $L_T(N) = L_T(N_1)$, we have $L_{T'}(N) = L_T(N) \cap X' = L_T(N_1) \cap X' = L_{T'}(N_1)$. Similarly, $L_T(N) = L_T(N_2)$ implies that $L_{T'}(N) = L_{T'}(N_2)$.
- When N represents $\sqcup$: If $L_T(N) = L_T(N_1)$, we have $L_{T'}(N) = L_T(N) \cap X' = L_T(N_1) \cap X' = L_{T'}(N_1)$. Otherwise, if $L_T(N) = L_T(N_2)$, we can prove similarly that $L_{T'}(N) = L_{T'}(N_2)$. Therefore, $L_{T'}(N) = L_{T'}(N_1)$ or $L_{T'}(N) = L_{T'}(N_2)$.
- When N represents $\odot$: Because $L_T(N) = L_T(N_1) \cup L_T(N_2)$, we have $L_{T'}(N) = L_T(N) \cap X' = (L_T(N_1) \cup L_T(N_2)) \cap X' = (L_T(N_1) \cap X') \cup (L_T(N_2) \cap X') = L_{T'}(N_1) \cup L_{T'}(N_2)$.
- When N represents $\otimes$: Similar to the above, we have $L_T(N) = L_{T'}(N_1) \cup L_{T'}(N_2)$. Also, $L_T(N_1) \cap L_T(N_2) = \emptyset$ indicates that $L_{T'}(N_1) \cap L_{T'}(N_2) = \emptyset$.

Therefore, T' is a satisfaction tree for ϕ. And since the root of T is labeled with X', X' satisfies ϕ according to Theorem 2.1.2.

According to the above argument, if ϕ is satisfiable, then there exists a set X' of no more than $|\phi|$ users and a configuration $\langle U, UR\rangle$, such that X' satisfies ϕ under $\langle U, UR\rangle$. Users not in X' can be removed from the configuration without affecting the satisfaction of ϕ. Also, those roles in UR that do not appear in ϕ can be removed too. Since there are no more than $|\phi|$ roles in ϕ and there are no more than $|\phi|$ users in X', we have $|UR| \leq |\phi|^2$. Therefore, the lemma holds.

Proof of Theorem 2.3.4

Since we have already proved that certain subcases of TSAT are $\mathbf{NP}$-hard, to prove the theorem, we just need to show that the problem is in $\mathbf{NP}$. Given a term ϕ, a nondeterministic Turing machine may guess a configuration $\langle U, UR\rangle$, a userset X, and a satisfaction tree T whose root is labeled with X. According to Lemma 2.3.3, the size of X and $\langle U, UR\rangle$ is

bounded by $|\phi|^2$. Also, according to Theorem 2.1.2, X satisfies ϕ if and only if there is a satisfaction tree of ϕ whose root is labeled with X. There are no more than $2|\phi| - 1$ nodes in T and the size of the set labeling a node is bounded by $|X|$. Therefore, the size of T is polynomial in the size of input. The Turing machine may verify whether T is a satisfaction tree by following the rules specified in Definition 2.1.4. It is clear that the verification can be done in polynomial time by following the structure of T. Therefore, TSAT is in $\mathbf{NP}$.

A.2.2 Proof of Lemma 2.3.7, Lemma 2.3.8, and Theorem 2.3.9

Proof of Lemma 2.3.7

Proof by induction on the structure of term ϕ.

Base case: When $\phi = r$ or $\phi = \mathsf{All}$, we have $C(\phi) = \{1\} \subseteq \{1, 2, \ldots, |\phi|\}$. Otherwise, when ϕ is in the form of ϕ_1^+ where ϕ_1 is a unit term, according to Definition 2.3.1, we have $C(\phi) = \{i | i \in [1, \infty)\} = W \cup \{|\phi| + 1, |\phi| + 2, \ldots\}$, where $W = \{1, 2, \ldots, |\phi|\}$.

Inductive case: When ϕ is in the form of $(\phi_1 \ \mathsf{op} \ \phi_2)$, assume that the lemma holds for ϕ_1 and ϕ_2. Let W_1 denote a subset of $\{1, 2, \ldots, |\phi_1|\}$ and W_2 denote a subset of $\{1, 2, \ldots, |\phi_2|\}$. We have the following three cases:

Case 1: Both $C(\phi_1)$ and $C(\phi_2)$ are finite. Let $C(\phi_1) = W_1$ and $C(\phi_2) = W_2$. Since $|\phi| = |\phi_1| + |\phi_2|$, it follows from Definition 2.3.1 that $C(\phi) \subseteq \{1, 2, \ldots, |\phi|\}$, because for any $c_1 \in C(\phi_1)$ and $c_2 \in C(\phi_2)$, $c_1 + c_2 \leq |\phi_1| + |\phi_2| = |\phi|$.

Case 2: Exactly one of $C(\phi_1)$ and $C(\phi_2)$ is an infinite set. Without loss of generality, assume that $C(\phi_1) = W_1$ and $C(\phi_2) = W_2 \cup \{|\phi_2| + 1, |\phi_2| + 2, \ldots\}$. We compute $C(\phi)$ according to op :

- $\mathsf{op} = \sqcup$: $C(\phi) = C(\phi_1) \cup C(\phi_2) = W_1 \cup W_2 \cup \{|\phi_2| + 1, |\phi_2| + 2, \ldots\} = W_1 \cup W_2 \cup \{|\phi_2| + 1, \ldots, |\phi|\} \cup \{|\phi| + 1, |\phi| + 2, \ldots\}$, in which $W_1 \cup W_2 \cup \{|\phi_2|, \ldots, |\phi|\}$ is a subset of $\{1, 2, \ldots, |\phi|\}$.
- $\mathsf{op} = \sqcap$: $C(\phi) = C(\phi_1) \cap C(\phi_2)$ is a subset of W_1, which is a subset of $\{1, 2, \ldots, |\phi|\}$.

- $\mathsf{op} = \odot$:

$$\begin{aligned} C(\phi) =& \{i \mid \exists c_1 \in W_1 \, \exists c_2 \in W_2 \, [max(c_1, c_2) \leq i \leq c_1 + c_2]\} \\ & \cup \{max(min(W_1), |\phi_2| + 1), max(min(W_1), |\phi_2| + 1) + 1, \dots\} \\ =& \{i \mid \exists c_1 \in W_1 \, \exists c_2 \in W_2 \, [max(c_1, c_2) \leq i \leq c_1 + c_2]\} \\ & \cup \{max(min(W_1), |\phi_2| + 2, \dots, |\phi|\} \cup \{|\phi| + 1, |\phi| + 2, \dots\} \end{aligned}$$

  Note that $\{i \mid \exists c_1 \in W_1 \, \exists c_2 \in W_2 \, [max(c_1, c_2) \leq i \leq c_1 + c_2]\} \cup \{max(min(W_1), |\phi_2| + 1), \dots, |\phi|\}$ is a subset of $\{1, 2, \dots, |\phi|\}$, as $c_1 + c_2 \leq |\phi_1| + |\phi_2| = |\phi|$.

- $\mathsf{op} = \otimes$:

$$\begin{aligned} C(\phi) =& \{c_1 + c_2 | c_1 \in W_1 \wedge (c_2 \in W_2 \vee c_2 \in [|\phi_2|, \infty))\} \\ =& \{c_1 + c_2 | c_1 \in W_1 \wedge c_2 \in W_2\} \\ & \cup \{min(W_1) + |\phi_2| + 1, min(W_1) + |\phi_2| + 2, \dots\} \\ =& \{c_1 + c_2 | c_1 \in W_1 \wedge c_2 \in W_2\} \\ & \cup \{min(W_1) + |\phi_2| + 1, \dots, |\phi|\} \cup \{|\phi| + 1, |\phi| + 2, \dots\} \end{aligned}$$

  Note that $\{c_1 + c_2 | c_1 \in W_1 \wedge c_2 \in W_2\} \cup \{min(W_1) + |\phi_2|, \dots, |\phi|]\}$ is a subset of $\{1, 2, \dots, |\phi|\}$.

Case 3: Both $C(\phi_1)$ and $C(\phi_2)$ are infinite sets, where $C(\phi_1) = W_1 \cup \{i | i \in [|\phi_1|, \infty)\}$ and $C(\phi_2) = W_2 \cup \{i | i \in [|\phi_2|, \infty)\}$. The argument is similar to Case 2. We omit the details here.

Proof of Lemma 2.3.8

When $\phi = \mathsf{All}$ or $\phi = r$ or $\phi = \phi_1^+$, $C(\phi)$ can be computed in constant time according to Definition 2.3.1.

There are $|\phi| - 1$ binary operators in ϕ. Hence, to prove the lemma, we just need to prove that, given $C(\phi_1)$ and $C(\phi_2)$, $C(\phi)$ can be computed in time polynomial in the size of $|\phi|$, where $\phi = (\phi_1 \text{ op } \phi_2)$.

According to Lemma 2.3.7, we may represent the characteristic set of a term ϕ as a tuple. Let $W \subseteq \{1, \ldots, |\phi|\}$. When $C(\phi) = W$, we represent $C(\phi)$ as a tuple $\langle |\phi|, W, 0\rangle$; when $C(\phi) = W \cup \{|\phi|+1, |\phi|+2, \ldots\}$, we represent $C(\phi)$ as a tuple $\langle |\phi|, W, 1\rangle$. In other words, the last element (either 0 or 1) of the tuple indicates whether $C(\phi)$ contains $\{|\phi|+1, |\phi|+2, \ldots\}$ or not.

Given $C(\phi_1)$ and $C(\phi_2)$, we represent them as tuples $\langle |\phi_1|, W_1, f_1\rangle$ and $\langle |\phi_2|, W_2, f_2\rangle$, where $f_1, f_2 \in \{0, 1\}$. Now, we show that computing the tuple-representation $\langle |\phi|, W, f\rangle$ of $C(\phi)$ can be done in polynomial time. Note that $|\phi| = |\phi_1| + |\phi_2|$. We just need to determine W and f.

- Case $f_1 = f_2 = 0$: According to Definition 2.3.1, it is clear that $f = 0$. Computing W from W_1 and W_2, by following Definition 2.3.1, involves set union/intersection or computing the sums of pairs of elements, which can be done in $O(|\phi_1| \cdot |\phi_2|)$.
- Case $f_1 = 0$ and $f_2 = 1$: According to Definition 2.3.1, if op $= \sqcap$, then $f = 0$; otherwise, $f = 1$. Computing W from W_1 and $W_2 \cup \{|\phi_2|+1, \ldots, |\phi|\}$ can be done in $O(|\phi_1| \cdot |\phi|)$.
- Case $f_1 = 1$ and $f_2 = 0$: Similar to the above.
- Case $f_1 = 1$ and $f_2 = 1$: According to Definition 2.3.1, we have $f = 1$. Computing W from $W_1 \cup \{|\phi_1|+1, \ldots, |\phi|\}$ and $W_2 \cup \{|\phi_2|+1, \ldots, |\phi|\}$ can be done in $O(|\phi|^2)$.

In summary, computing $C(\phi)$ takes polynomial time.

Proof of Theorem 2.3.9

Given a term ϕ, let $C'(\phi)$ be the set of all integers k's such that there is a userset of size k that satisfies ϕ under some configuration. We would like to prove that $C'(\phi) \equiv C(\phi)$. We prove this by induction on the structure of ϕ.

Base case: when $\phi = \mathsf{All}$, ϕ is satisfied by any userset that is singleton; when $\phi = r$, ϕ is satisfied by a singleton containing a user who is a member of r. Hence, we have $C'(\mathsf{All}) = C'(r) = \{1\}$. According to Definition 2.3.1, $C'(\phi) \equiv C(\phi)$.

Inductive case: assume that $C'(\phi) \equiv C(\phi)$ when $|\phi| < k$, where $|\phi|$ is the number of atomic terms in ϕ. When $|\phi| = k$, we have:

- Case $\phi = \phi_1 \sqcup \phi_2$: It follows from the definition of satisfaction (Definition 2.1.3) that $C'(\phi_1 \sqcup \phi_2) = C'(\phi_1) \cup C'(\phi_2)$. By inductive assumption, $C'(\phi_1) \equiv C(\phi_1)$ and $C'(\phi_2) \equiv C(\phi_2)$. According to Definition 2.3.1, we have $C'(\phi) \equiv C(\phi)$.

- Case $\phi = \phi_1 \sqcap \phi_2$: It follows from Definition 2.1.3 that $C'(\phi_1 \sqcap \phi_2) \subseteq C'(\phi_1) \cap C'(\phi_2)$. In the following, we prove that $C'(\phi_1 \sqcap \phi_2) \supseteq C'(\phi_1) \cap C'(\phi_2)$, where ϕ_1 and ϕ_2 are free of negation and explicit sets of users.

  Assume that X_1 is a size-k userset that satisfies ϕ_1 under configuration $\langle U, UR_1 \rangle$ and X_2 is a size-k userset that satisfies ϕ_2 under configuration $\langle U, UR_2 \rangle$. Since ϕ_1 and ϕ_2 do not contain explicit sets of users, the names of users are not important. Hence, we can assume that $X_1 = X_2$. Also, since ϕ_1 does not contain negation, X_1 still satisfies ϕ_1 even if we assign more roles to users in X_1. Therefore, X_1 satisfies ϕ_1 under $\langle U, UR_1 \cup UR_2 \rangle$. Also, X_1 (which is equivalent to X_2) satisfies ϕ_2 under $\langle U, UR_1 \cup UR_2 \rangle$. Therefore, X_1 satisfies $\phi_1 \sqcap \phi_2$. Since $|X_1| = k$, we have $k \in C'(\phi_1 \sqcap \phi_2)$. Hence, $C'(\phi_1 \sqcap \phi_2) \supseteq C'(\phi_1) \cap C'(\phi_2)$.

  In summary, we have $C'(\phi_1 \sqcap \phi_2) = C'(\phi_1) \cap C'(\phi_2)$. By inductive assumption, $C'(\phi_1) \equiv C(\phi_1)$ and $C'(\phi_2) \equiv C(\phi_2)$. According to Definition 2.3.1, we have $C'(\phi) \equiv C(\phi)$.

- Case $\phi = \phi_0^+$: It follows from the computation of $C'(\mathsf{All}), C'(r), C'(\phi_1 \sqcup \phi_2)$ and $C'(\phi_1 \sqcap \phi_2)$ that $C'(\phi_0) = \{1\}$, where ϕ_0 is a unit term free of explicit sets of users and negation. Given a configuration $\langle U, UR \rangle$ and a singleton $\{u_1\}$ such that $\{u_1\}$ satisfies ϕ_0, we create $u_2, \ldots, u_n$ such that u_i ($i \in [2, n]$) is assigned to precisely the same set of roles as u_1. In this case, $\{u_1, \ldots, u_n\}$ satisfies ϕ_0^+. In other words, ϕ_0^+ may be satisfied by n users for any $n \geq 1$. That is to say, $C'(\phi_0^+) = \{i \mid i \in [1, \infty)\}$. According to Definition 2.3.1, we have $C'(\phi) \equiv C(\phi)$.

- Case $\phi = \phi_1 \odot \phi_2$: Let X be a userset that satisfies $(\phi_1 \odot \phi_2)$. There exist X_1 and X_2 such that X_1 satisfies ϕ_1, X_2 satisfies ϕ_2, and $X_1 \cup X_2 = X$. By the definition of C', there exist $c_1 \in C'(\phi_1)$ and $c_2 \in C'(\phi_2)$ such that $|X_1| = c_1$ and $|X_2| = c_2$. Hence, $max(c_1, c_2) \leq |X| \leq c_1 + c_2$.

Given $c_1 \in C'(\phi_1)$ and $c_2 \in C'(\phi_2)$, there exist X_1 and X_2 such that X_1 satisfies ϕ_1 under $\langle U_1, UR_1\rangle$, X_2 satisfies ϕ_2 under $\langle U_2, UR_2\rangle$, $|X_1| = c_1$ and $|X_2| = c_2$. For any integer $k \in [max(c_1, c_2), c_1 + c_2]$, we may name users in such a way that $|X_1 \cap X_2| = c_1 + c_2 - k$. In this case, $X = X_1 \cup X_2$ satisfies $(\phi_1 \odot \phi_2)$ under $\langle U_1 \cup U_2, UR_1 \cup UR_2\rangle$ and $|X| = k$.

In summary, $C'(\phi_1 \odot \phi_2) = \{\ i \mid \exists\ c_1 \in C'(\phi_1) \quad \exists\ c_2 \in C'(\phi_2)\ [\ max(c_1, c_2) \leq i \leq c_1 + c_2\ ]\}$. By inductive assumption, $C'(\phi_1) \equiv C(\phi_1)$ and $C'(\phi_2) \equiv C(\phi_2)$. According to Definition 2.3.1, we have $C'(\phi) \equiv C(\phi)$.

- Case $\phi = \phi_1 \otimes \phi_2$: On the one hand, userset X satisfies $(\phi_1 \otimes \phi_2)$ if and only if there exist X_1 and X_2 such that $X_1 \cup X_2 = X$, $X_1 \cap X_2 = \emptyset$ and X_1, X_2 satisfy ϕ_1, ϕ_2 respectively. By definition of C', we have $|X_1| \in C'(\phi_1)$ and $|X_2| \in C'(\phi_2)$. Therefore, $|X| = (|X_1| + |X_2|) \in \{\ c_1 + c_2 \mid c_1 \in C'(\phi_1)\ \wedge\ c_2 \in C'(\phi_2)\ \}$.

  On the other hand, given any $c_1 \in C'(\phi_1)$ and $c_2 \in C'(\phi_2)$, by definition of C', there exist X_1 and X_2 that satisfy ϕ_1 and ϕ_2 under $\langle U_1, UR_1\rangle$ and $\langle U_1, UR_1\rangle$ respectively, such that $|X_1| = c_1$ and $|X_2| = c_2$. Name the users in such a way that $X_1 \cap X_2 = \emptyset$. We have $X = X_1 \cup X_2$ satisfies $(\phi_1 \otimes \phi_2)$ under $\langle U_1 \cup U_2, UR_1 \cup UR_2\rangle$, where $|X| = |X_1| + |X_2| = c_1 + c_2$.

  In summary, $C'(\phi_1 \otimes \phi_2) = \{\ c_1 + c_2 \mid c_1 \in C'(\phi_1)\ \wedge\ c_2 \in C'(\phi_2)\ \}$. By inductive assumption, $C'(\phi_1) \equiv C(\phi_1)$ and $C'(\phi_2) \equiv C(\phi_2)$. According to Definition 2.3.1, we have $C'(\phi) \equiv C(\phi)$.

In conclusion, we have $C'(\phi) \equiv C(\phi)$ and Theorem 2.3.9 holds.

A.3 Proofs of Theorems in Section 2.4

In the following proofs, $(\mathsf{op}_k\phi)$ denotes k copies of ϕ connected together by operator op and $(\mathsf{op}_{i=1}^{n} r_i)$ denotes $(r_1\ \mathsf{op} \ldots \mathsf{op}\ r_n)$. Given $R = \{r_1, \ldots, r_m\}$, $(\mathsf{op}R)$ denotes $(r_1\ \mathsf{op} \ldots \mathsf{op}\ r_m)$.

A.3.1 The Five Intractability Subcases of UTS

Lemma A.3.1 UTS $\langle \sqcup, \odot \rangle$ is **NP**-hard.

Proof We use a reduction from the **NP**-complete SET COVERING problem [25]. In the SET COVERING problem, we are given a finite set $S = \{e_1, \ldots, e_n\}$, a family of S's subsets $F = \{S_1, \ldots, S_m\}$, and an integer $k < m$, and we ask whether there exists a sub-family of sets $F' \subseteq F$ whose union is S and $|F'| \leq k$. Given such an instance, our reduction maps each element in S to a user and to a role. We construct a configuration $\langle U, UR \rangle$ such that $U = \{u_1, \ldots, u_n\}$ and $UR = \{(u_i, r_i) \mid i \in [1, n]\}$, and a term $\phi = (\bigodot_k (\bigsqcup_{i=1}^{m} (\bigodot R_i)))$, where R_i is a set of roles such that $r_j \in R_i$ if and only if $e_j \in S_i$.

We now demonstrate that U satisfies ϕ under $\langle U, UR \rangle$ if and only if there exist k sets in F whose union is S. On the one hand, assume that U satisfies ϕ, by definition. U has k subsets $U_1, \ldots, U_k$ such that $\bigcup_{i=1}^{k} U_i = U$ and every U_i satisfies $(\bigsqcup_{i=1}^{m} (\bigodot R_i))$. U_i satisfies $(\bigsqcup_{i=1}^{m} (\bigodot R_i))$ if and only if U_i satisfies a certain $(\bigodot R_{x_i})$, where $x_i \in [1, m]$. From the construction of R_{x_i}, U_i satisfies $(\bigodot R_{x_i})$ if and only if $U_i = \{u_a \mid e_a \in S_{x_i}\}$. Since $\bigcup_{i=1}^{k} U_i = U$, we have $\bigcup_{i=1}^{k} S_{x_i} = S$. The answer to the set covering problem is "yes".

On the other hand, assume that there are k sets in F whose union is S. Without loss of generality, we assume that $\bigcup_{i=1}^{k} S_i = S$. In this case, we divide U into k sets $U_1, \ldots, U_k$ such that $U_i = \{u_j \mid e_j \in S_i\}$. Since $\bigcup_{i=1}^{k} S_i = S$, we have $\bigcup_{i=1}^{k} U_i = U$. Furthermore, since $U_i = \{u_j \mid e_j \in S_i\}$, from the construction of R_i, we have U_i satisfies $(\bigodot R_i)$ for every $i \in [1, k]$. Therefore, U satisfies $\phi = (\bigodot_k (\bigsqcup_{i=1}^{m} (\bigodot R_i)))$. ■

Lemma A.3.2 UTS $\langle \sqcap, \odot \rangle$ is **NP**-hard.

Proof We use a reduction from the **NP**-complete SET COVERING problem [25]. Given $S = \{e_1, \ldots, e_n\}$, a family of S's subsets $F = \{S_1, \ldots, S_m\}$, and an integer $k < m$, our reduction maps each element $e_j \in S$ to a role r_j and each $S_i \in F$ to a user u_i. We construct a configuration $\langle U, UR \rangle$ such that $U = \{u_1, \ldots, u_m\}$ and $UR = \{(u_i, r_j) \mid e_j \in S_i\}$, and a term $\phi = (((\bigodot_k \mathsf{All}) \sqcap (\bigodot_{i=1}^{n} r_i)) \odot (\bigodot_m \mathsf{All}))$.

We now demonstrate that U satisfies ϕ under $\langle U, UR \rangle$ if and only if there exist k sets in family F whose union is S. On the one hand, assume that U satisfies ϕ. Since $(\bigodot_m \mathsf{All})$

can be satisfied by any nonempty userset with no more than m users, U always satisfies $(\bigodot_m \mathsf{All})$ and it satisfies ϕ if and only if there is $U' \subseteq U$ such that U' satisfies $((\bigodot_k \mathsf{All}) \sqcap (\bigodot_{i=1}^{n} r_i))$. U' satisfying $(\bigodot_k \mathsf{All})$ indicates that $|U'| \leq k$, while U' satisfying $(\bigodot_{i=1}^{n} r_i)$ indicates that users in U' together have membership of all roles in $\{r_1, \ldots, r_n\}$. Without loss of generality, suppose $U' = \{u_1, \ldots, u_t\}$, where $t \leq k$. Because $(u_i, r_j) \in UR$ if and only if $e_j \in S_i$, the union of $\{S_1, \ldots, S_t\}$ is S. The answer to the SET COVERING problem is "yes".

On the other hand, assume that k subsets in F cover S. Without loss of generality, we assume that $\bigcup_{i=1}^{k} S_i = S$. From the construction of UR, users $u_1, \ldots, u_k$ together have membership of all roles in $\{r_1, \ldots, r_n\}$. In this case, $\{u_1, \ldots, u_k\}$ satisfies $(\bigodot_{i=1}^{n} r_i)$. Also, $\{u_1, \ldots, u_k\}$ satisfies $(\bigodot_k \mathsf{All})$. Hence, $\{u_1, \ldots, u_k\}$ satisfies $((\bigodot_k \mathsf{All}) \sqcap (\bigodot_{i=1}^{n} r_i))$. $(\bigodot_m \mathsf{All})$ is also satisfied by U. Therefore, U satisfies ϕ. ■

Lemma A.3.3 UTS $\langle \odot, \otimes \rangle$ is **NP**-hard.

Proof We use a reduction from the **NP**-complete DOMATIC NUMBER problem [25]. Given a graph $G(V, E)$, the Domatic Number problem asks whether V can be partitioned into k disjoint nonempty sets $V_1, V_2, \ldots, V_k$, such that each V_i is a dominating set for G. V' is a dominating set for $G = (V, E)$ if for every node u in $V - V'$, there is a node v in V' such that $(u, v) \in E$.

Given a graph $G = (V, E)$ and a threshold k, let $U = \{u_1, u_2, \ldots, u_n\}$ and $R = \{r_1, r_2, \ldots, r_n\}$, where n is the number of nodes in V. Each user in U corresponds to a node in G, and $v(u_i)$ denotes the node corresponding to user u_i. $UR = \{(u_i, r_j) \mid i = j \text{ or } (v(u_i), v(u_j)) \in E\}$. Let $\phi = (\bigotimes_k (\bigodot_{i=1}^{n} r_i))$.

A dominating set in G corresponds to a set of users that together have membership of all the n roles. U satisfies ϕ under $\langle U, UR \rangle$ if and only if U can be divided into k pairwise disjoint sets, each of which has role membership of $r_1, r_2, \ldots, r_n$. Therefore, the answer to the Domatic Number problem is "yes" if and only if U satisfies ϕ under $\langle U, UR \rangle$. ■

Lemma A.3.4 UTS $\langle \otimes, \sqcup \rangle$ is **NP**-hard.

Proof We use a reduction from the **NP**-complete SET PACKING problem [25], which asks: Given a finite set $S = \{e_1, \ldots, e_n\}$, a family of S's subsets $F = \{S_1, \ldots, S_m\}$, and an integer k, whether there are k pairwise disjoint elements (which are sets) in F? Without loss of generality, we assume that $S_i \not\subseteq S_j$ when $i \neq j$. (If $S_i \subseteq S_j$, one can remove S_j without affecting the answer.) Let $U = \{u_0, u_1, \ldots, u_n\}$, $R = \{r_1, \ldots, r_n\}$ and $UR = \{(u_i, r_i) \mid 1 \leq i \leq n\}$. Note that u_0 is a user that is not assigned to any role. We then construct a term $\phi = ((\bigotimes_k (\bigsqcup_{i=1}^m (\bigotimes R_j))) \otimes \phi_{nonempty})$, where $R_j = \{r_i \mid e_i \in S_j\}$ and $\phi_{nonempty} = (\mathsf{All} \sqcup (\mathsf{All} \otimes \mathsf{All}) \sqcup \cdots \sqcup (\bigotimes_m \mathsf{All}))$.

We show that U satisfies ϕ under $\langle U, UR \rangle$ if and only if there are k pairwise disjoint elements in family F. As the only member of r_i is u_i, the only userset that satisfies $\phi_i = (\bigotimes R_j)$ is $U_j = \{u_i \mid e_i \in S_j\}$. Hence, a userset X satisfies $\phi' = (\bigsqcup_{i=1}^m \phi_i)$ if and only if X equals to some U_j.

Without loss of generality, assume that $S_1, \ldots, S_k$ are k pairwise disjoint sets. Then, $U_1, \ldots, U_k$ are k pairwise disjoint sets of users. U_1 satisfies ϕ_1, and thus satisfies ϕ'. Similarly, we have U_i satisfies ϕ' for every i from 1 to k. Furthermore, since $u_0 \notin U_i$ for any $i \in [1, k]$, we have $\bigcup_{i=1}^k U_i \subset U$. Hence, U can be divided into two nonempty subset $\bigcup_{i=1}^k U_i$ and $U' = U - \bigcup_{i=1}^k U_i$ such that $\bigcup_{i=1}^k U_i$ satisfies $(\bigotimes_k (\bigsqcup_{i=1}^m (\bigotimes R_j)))$ and U' satisfies $\phi_{nonempty}$. In other words, U satisfies ϕ.

On the other hand, suppose that U satisfies ϕ. Then, U has a strict subset U' with $u_0 \notin U'$, such that U' can be divided into k pairwise disjoint sets $\hat{U_1}, \ldots, \hat{U_k}$, such that each $\hat{U_i}$ satisfies ϕ'. In order to satisfy ϕ', $\hat{U_i}$ must satisfy a certain ϕ_{a_i} and hence be equivalent to U_{a_i}, where $a_i \in [1, m]$. The assumption that $\hat{U_1}, \ldots, \hat{U_k}$ are pairwise disjoint indicates that $U_{a_1}, \ldots, U_{a_k}$ are also pairwise disjoint. Therefore, their corresponding sets $S_{a_1}, \ldots, S_{a_k}$ are pairwise disjoint. The answer to the SET PACKING problem is "yes". ■

Lemma A.3.5 UTS $\langle \sqcap, \otimes \rangle$ is **NP**-hard.

Proof We use a reduction from the NP-complete SET COVERING problem, which asks: Given a family $F = \{S_1, \ldots, S_m\}$ of subsets of a finite set $S = \{e_1, \ldots, e_n\}$ and an integer

k no larger than m, whether there is a subfamily of sets $F' \leq F$ whose union is S and $|F'| \leq k$?

Given S and F, let $U = \{u_1, u_2, \ldots, u_m\}$, $R = \{r_1, r_2, \ldots, r_n\}$ and $UR = \{(u_i, r_j) \mid e_j \in S_i\}$. Let $\phi = ((\bigsqcap_{i=1}^{n} \left(r_i \otimes \left(\bigotimes_{k-1} \mathsf{All}\right)\right)) \otimes (\bigotimes_{m-k} \mathsf{All}))$. We now demonstrate that U satisfies ϕ under $\langle U, UR\rangle$ if and only if there are k sets in family F whose union is S. Without loss of generality, assume that $k < m$.

First, assume that U satisfies ϕ. Since $(\bigotimes_{m-k} \mathsf{All})$ can be satisfied by any userset with $m - k$ users, U satisfies ϕ if and only if there is a size-k subset U' of U that satisfies $\left(r_i \otimes \left(\bigotimes_{k-1} \mathsf{All}\right)\right)$ for every $i \in [1, n]$. This means that users in U' together have membership of all roles in $\{r_1, \ldots, r_n\}$. Suppose $U' = \{u_{a_1}, \ldots, u_{a_k}\}$, where $a_i \in [1, m]$. Because $(u_i, r_j) \in UR$ if and only if $e_j \in S_i$, the union of $\{S_{a_1}, \ldots, S_{a_k}\}$ is S. The answer to the Set Covering problem is "yes".

Second, without loss of generality, assume that $\bigcup_{i=1}^{k} S_i = S$. From the construction of UR, users $u_1, \ldots, u_k$ together have membership of $r_1, \ldots, r_n$. In this case, $\{u_1, \ldots, u_k\}$ satisfies $\left(r_i \otimes \left(\bigotimes_{k-1} \mathsf{All}\right)\right)$ for every $i \in [1, n]$. Since $k < m$, $\{u_1, \ldots, u_k\}$ is a strict subset of U. Therefore, U can be divided into two nonempty subset $\{u_1, \ldots, u_k\}$ and $U - \{u_1, \ldots, u_k\}$ such that $\{u_1, \ldots, u_k\}$ satisfies $(\bigsqcap_{i=1}^{n} \left(r_i \otimes \left(\bigotimes_{k-1} \mathsf{All}\right)\right))$ and $U - \{u_1, \ldots, u_k\}$ satisfies $(\bigotimes_{m-k} \mathsf{All})$. In other words, U satisfies ϕ. ■

A.3.2 Proof that UTS Is in NP

Lemma A.3.6 UTS $\langle \neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ is in **NP**.

Proof Given a term ϕ, a configuration $\langle U, UR\rangle$ and a userset X, according to Theorem 2.1.2, X satisfies ϕ if and only if there exists a satisfaction tree of ϕ whose root is labeled with X. A non-deterministic Turing machine may guess a satisfaction tree T of ϕ such that the root of T is labeled with X. From the proof of Theorem 2.3.4, the size of T is polynomial in the size of ϕ and verifying whether T is a satisfaction tree can be done in polynomial time by following the rules in Definition 2.1.4. Therefore, UTS $\langle \neg, +, \sqcup, \sqcap, \odot, \otimes\rangle$ is in **NP**. ■

A.3.3 The Tractable Cases

Lemma A.3.7 UTS for 4CF terms is in **P**.

Proof Given a 4CF term $\phi = (P_1 \odot \cdots \odot P_n)$, where for each k such that $1 \leq k \leq n$, P_k is a 3CF term of the form $(\phi_{k,1} \otimes \phi_{k,2} \otimes \cdots \otimes \phi_{k,m_k})$, and each $\phi_{k,j}$ is a 1CF term. Let $t_{k,j}$ be the base (which is a unit term) of $\phi_{k,j}$. $T_k = \{t_{k,1}, t_{k,2}, \ldots, t_{k,m_k}\}$ is a multiset of the base of the 1CF terms in P_k.

Given a userset $X = \{u_1, \ldots, u_n\}$ and configuration $\langle U, UR\rangle$, we present an algorithm that determines whether X satisfies ϕ under $\langle U, UR\rangle$.

Step 1 The first step checks that each P_k is satisfied by some subset of X. For each k such that $1 \leq k \leq n$, do the following. Construct a bipartite graph $G(X, T_k)$, in which one partition consists of users in X and the other consists of all the $t_{k,j}$'s in T_k; and there is an edge between $u \in X$ and $t_{k,j}$ if and only if $\{u\}$ satisfies $t_{k,j}$. Compute a maximal matching of the graph $G(X, T_k)$, if the size of the matching is less than m_k, returns "no", as this means that X does not contain a subset that satisfies P_k; thus X does not satisfy ϕ.

Step 2 The second step checks that each user in X can be "consumed" by some unit term in ϕ. Let $G(A, B)$ denote the bipartite graph in which one partition, A, consists of users in X, and the other partition, B, consists of all the $t_{k,j}$'s in $T_1 \cup T_2 \cup \cdots \cup T_n$. Furthermore, for any unit term t that occurs as t^+ in ϕ, we make sure that B has at least $|X|$ copies of t by adding additional copies of t if necessary. There is an edge between $u \in A$ and $t \in B$ if and only if $\{u\}$ satisfies t. Compute a maximal matching of the graph $G(X, T)$, if the matching has size less than $|X|$, returns "no".

Step 3 Return "yes".

It is not difficult to see that if the algorithm returns "no", then X does not satisfy ϕ. We now show that if the algorithm returns "yes", then X satisfies ϕ. If the algorithm returns "yes", then for each k, the graph $G(X, T_k)$ has a matching of size m_k. Let X_k be the set of users involved in the matching. X_k satisfies P_k. Let $X' = X_1 \cup X_2 \cup \cdots \cup X_n$. If $X' = X$, then clearly X satisfies ϕ. If $X' \subset X$, then find a user u in $X \setminus X'$, and do the following: Find the term t that is matched with u in the maximal matching computed in

step 2. Such a term must exist, since the matching has size $|X|$. Without loss of generality, assume that t appears in P_1, and X_1 contains a user w that is matched with t; then change X_1 by replacing w with u. Clearly, the new X_1 still satisfies P_1. Compute X' again, and if $X' \subset X$, find another user in $X \setminus X'$ and repeat the previous process. Note that X' will grow if w appears in some other X_k. Also observe that, the newly added matching between u and t will never be removed again in future, because no other user is matched with t in the maximal matching computed in step 2; as a result, u will always remain in X'. Therefore, after each step, one new user will be added to X' and will never be removed. After at most $|X|$ steps, we will have $X' = X$. ■

A.4 Proof of Theorem 2.5.1

Proofs of the P results in Theorem 2.5.1

We first prove the following lemma, which will be useful.

Lemma A.4.1 The following properties hold.

1. A userset X satisfies a unit term t if and only if X is a singleton and the only user in X satisfies t.
2. A userset X satisfies a term t^+, where t is a unit term, if and only if every user in X satisfies t.
3. If a userset X satisfies a term ϕ that is built using only $\neg, +, \sqcap, \sqcup$, then every user in X satisfies ϕ.
4. A userset X is safe with respect to a 1CF term ϕ if and only if there exists a user in X that satisfies t.

Proof Properties 1 and 2 follow from the definition of term satisfaction. Observe that a unit term can be satisfied only by a singleton.

Property 3. The term ϕ can be decomposed into subterms in 1CF form, connected using $\sqcap$ and $\sqcup$. By definition, X satisfies $\phi_1 \sqcap \phi_2$ if and only if X satisfies both ϕ_1 and ϕ_2, and X satisfies $\phi_1 \sqcup \phi_2$ if and only if X satisfies either ϕ_1 or ϕ_2. Identify all 1CF subterms that X

satisfies, it follows from Properties 1 and 2 that each user in X satisfies all these subterms. Therefore, each user satisfies ϕ.

Property 4. For the "if" direction, if X contains a user u that satisfies t, then $\{u\}$ satisfies the term ϕ, and thus X is safe with respect to ϕ. For the "only if" direction, if X is safe with respect to ϕ, then X contains a subset X_0 that satisfies ϕ. Any user in X_0 must satisfy t according to Properties 1 and 2. ■

Lemma A.4.2 SAFE $\langle \neg, +, \sqcup, \odot \rangle$ is in $\mathbf{P}$.

Proof A userset X is safe with respect to $(\phi_1 \sqcup \phi_2)$ if and only if either X is safe with respect to ϕ_1 or X is safe with respect to ϕ_2. Furthermore, X is safe with respect to $(\phi_1 \odot \phi_2)$ if and only if X is safe with respect to both ϕ_1 and ϕ_2. Therefore, one can determine whether U is safe with respect to ϕ, which is built using only the operators in $\{\neg, +, \sqcup, \odot\}$, by following the structure of the term until reaching subterms in 1CF. From Property 4 of Lemma A.4.1, checking whether U is safe with respect to such a term amounts to checking whether there exists a user in U that satisfies t, which can be done in polynomial time. ■

Lemma A.4.3 SAFE $\langle \neg, +, \sqcup, \sqcap \rangle$ is in $\mathbf{P}$.

Proof Given a term ϕ which is built using only operators in $\{\neg, +, \sqcup, \sqcap\}$, we prove that a userset X is safe with respect to ϕ if and only if there exists a user $u \in X$ such that u satisfies ϕ. The "if" direction follows by definition. For the "only if" direction: Suppose that X contains a nonempty subset X_0 that satisfies ϕ, then by Property 3 of Lemma A.4.1, every user in X_0 satisfies ϕ; thus X must contain a user that satisfies ϕ. Therefore, to determine whether X is safe with respect to ϕ, one can, for each user in X, check whether the user satisfies ϕ. Checking whether one user satisfies a term using only operators in $\{\neg, +, \sqcup, \sqcap\}$ can be done in polynomial time. ■

Lemma A.4.4 SAFE $\langle \neg, +, \otimes \rangle$ is in $\mathbf{P}$.

Proof Given a term ϕ which does not contain any binary operator but $\otimes$, we show that determining whether a userset X is safe with respect to ϕ under a configuration $\langle U, UR \rangle$

can be reduced to the maximum matching problem on bipartite graphs, which can be solved in $O(MN)$ time, where M is the number of edges and N is the number of nodes in G [50].

Let s be the number of 1CF terms in ϕ and $t = |X|$. Since $\otimes$ is associative, ϕ can be equivalently expressed as $(\phi_1 \otimes \phi_2 \otimes \cdots \otimes \phi_s)$, where each ϕ_i is a 1CF term . Let $X = \{u_1, \ldots, u_t\}$. We construct a bipartite graph $G(V_1 \cup V_2, E)$, where each node in V_1 corresponds to a 1CF term in ϕ and each node in V_2 corresponds to a user in X. More precisely, $V_1 = \{a_1, \ldots, a_s\}$, $V_2 = \{b_1, \ldots, b_t\}$, and $(a_i, b_j) \in E$ if and only if $\{u_j\}$ satisfies ϕ_i. The resulting graph G has $s+t$ nodes and $O(st)$ edges, and can be constructed in time polynomial in the size of G. Solving the maximal matching problem for G takes time $O((s+t)st)$.

We now show that X is safe with respect to ϕ if and only if the maximal matching in the graph G has size s. If the maximal matching has size s, then each node in V_1 matches to a certain node in V_2, which means that the s 1CF terms in ϕ are satisfied by s distinct users in X; thus X contains a subset that satisfies ϕ. If X is safe with respect to ϕ, by definition, there exist s disjoint subsets $X_1, \ldots, X_s$ such that X_i $(i \in [1, s])$ satisfies ϕ_i and $\bigcup_{j=1}^{s} X_j \subseteq X$. From our construction of G, we may match a node corresponding to a user in X_i to the node corresponding to ϕ_i. In this case, a maximal matching of size s exists. ■

Proving the NP-completeness results in Table 2.2

Lemma A.4.5 $SAFE\langle\sqcap, \odot\rangle$ is **NP**-hard.

Proof We use a reduction from the **NP**-complete SET COVERING problem [25]. In the SET COVERING problem, we are given a family $F = \{S_1, \ldots, S_m\}$ of subsets of a finite set $S = \{e_1, \ldots, e_n\}$ and an integer k no larger than m, and we ask whether there is a subfamily of sets $F' \subseteq F$ whose union is S and $|F'| \leq k$.

Given S and F, we construct a configuration $\langle U, UR\rangle$ such that $(u_i, r_j) \in UR$ if and only if $e_j \in S_i$. Let $U = \{u_1, \ldots, u_m\}$ and $\phi = ((\bigodot_k \mathsf{All}) \sqcap (\bigodot_{i=1}^{n} r_i))$.

We now demonstrate that U is safe with respect to ϕ under $\langle U, UR\rangle$ if and only if there are no more than k sets in family F whose union is S.

First, if U is safe with respect to ϕ, by definition, a subset U' of U satisfies both $(\bigodot_k \mathsf{All})$ and $(\bigodot_{i=1}^{n} r_i)$. U' satisfying $(\bigodot_k \mathsf{All})$ indicates that $|U'| \leq k$, while U' satisfying $(\bigodot_{i=1}^{n} r_i)$ indicates that users in U' together have membership of r_i for every $i \in [1, n]$. Without loss of generality, suppose $U' = \{u_1, \ldots, u_t\}$, where $t \leq k$. Since $(u_i, r_j) \in UR$ if and only if $e_j \in S_i$, the union of $\{S_1, \ldots, S_t\}$ is S. The answer to the SET COVERING problem is "yes".

Second, without loss of generality, assume that $\bigcup_{i=1}^{k} S_i = S$. From the construction of UR, users $u_1, \ldots, u_k$ together have membership of r_i for every $i \in [1, n]$, which indicates that $\{u_1, \ldots, u_k\}$ is safe with respect to $(\bigodot_{i=1}^{n} r_i)$. Also, any non-empty subset of $\{u_1, \ldots, u_k\}$ satisfies $(\bigodot_k \mathsf{All})$. Hence, U is safe with respect to ϕ. ■

Lemma A.4.6 $SAFE\langle\odot, \otimes\rangle$ is **NP**-hard.

Proof We use a reduction from the **NP**-complete DOMATIC NUMBER problem [25]. Given a graph $G(V, E)$, the Domatic Number problem asks whether V can be partitioned into k disjoint sets $V_1, V_2, \ldots, V_k$, such that each V_i is a dominating set for G. V' is a dominating set for $G = (V, E)$ if for every node u in $V - V'$, there is a node v in V' such that $(u, v) \in E$.

Given a graph $G = (V, E)$ and an integer k, let $U = \{u_1, u_2, \ldots, u_n\}$ and $R = \{r_1, r_2, \ldots, r_n\}$, where n is the number of nodes in V. Each user in U corresponds to a node in G, and $v(u_i)$ denotes the node corresponding to user u_i. Let $UR = \{(u_i, r_j) \mid i = j \text{ or } (v(u_i), v(u_j)) \in E\}$ and $\phi = (\bigotimes_k (\bigodot_{i=1}^{n} r_i))$.

A dominating set in G corresponds to a set of users who together have membership of all the n roles. U is safe with respect to ϕ if and only if U has a subset U' that can be divided into k pairwise disjoint sets, each of which have role membership of $r_1, r_2, \ldots, r_n$. Therefore, the answer to the Domatic Number problem is "yes" if and only if U is safe with respect to ϕ. ■

Lemma A.4.7 $SAFE\langle\otimes, \sqcup\rangle$ is **NP**-hard.

Proof We use a reduction from the **NP**-complete SET PACKING problem [25], which asks, given a family $F = \{S_1, \ldots, S_m\}$ of subsets of a finite set $S = \{e_1, \ldots, e_n\}$ and an

integer k, whether there are k pairwise disjoint sets in family F. Without loss of generality, we assume that $S_i \not\subseteq S_j$ if $i \neq j$.

Given S and F, let $U = \{u_1, \ldots, u_n\}$, $R = \{r_1, \ldots, r_n\}$ and $UR = \{(u_i, r_i) \mid 1 \leq i \leq n\}$. We then construct a term $\phi = (\bigotimes_k (\bigsqcup_{i=1}^{m} (\bigotimes R_j)))$, where $R_j = \{r_i \mid e_i \in S_j\}$. We show that U is safe with respect to ϕ under $\langle U, UR\rangle$ if and only if there are k pairwise disjoint sets in family F.

As the only member of r_i is u_i, the only userset that satisfies $\phi_i = (\bigotimes R_j)$ is $U_j = \{u_i \mid e_i \in S_j\}$. A userset X satisfies $\phi' = (\bigsqcup_{i=1}^{m} \phi_i)$ if and only if X equals to some U_j.

First, without loss of generality, assume that $S_1, \ldots, S_k$ are k pairwise disjoint sets. Then, $U_1, \ldots, U_k$ are k pairwise disjoint sets of users. U_1 satisfies ϕ_1, and thus satisfies ϕ'. Similarly, U_i satisfies ϕ' for every i from 1 to k. Since $U_i \subseteq U$, U is safe with respect to ϕ.

Second, suppose U is safe with respect to ϕ. Then, U has a subset U' that can be divided into k pairwise disjoint sets $\hat{U_1}, \ldots, \hat{U_k}$, such that $\hat{U_i}$ satisfies ϕ_i. In order to satisfy ϕ', $\hat{U_i}$ must satisfy a certain ϕ_{a_i} and hence be equivalent to U_{a_i}. The assumption that $\hat{U_1}, \ldots, \hat{U_k}$ are pairwise disjoint indicates that $U_{a_1}, \ldots, U_{a_k}$ are also pairwise disjoint. Therefore, their corresponding sets $S_{a_1}, \ldots, S_{a_k}$ are pairwise disjoint. The answer to the Set Packing problem is "yes". ■

Lemma A.4.8 *SAFE* $\langle \sqcap, \otimes \rangle$ *is* **NP***-hard.*

Proof We use a reduction from the NP-complete SET COVERING problem, which asks, given a family $F = \{S_1, \ldots, S_m\}$ of subsets of a finite set $S = \{e_1, \ldots, e_n\}$ and an integer k no larger than m, whether there is a subfamily of sets $F' \subseteq F$ whose union is S and $|F'| \leq k$.

Given S and F, let $U = \{u_1, u_2, \ldots, u_m\}$, $R = \{r_1, r_2, \ldots, r_n\}$ and $UR = \{(u_i, r_j) \mid e_j \in S_i\}$. Let $\phi = (\bigsqcap_{i=1}^{n} \left(r_i \otimes \left(\bigotimes_{k-1} \mathsf{All}\right)\right))$. We now demonstrate that U satisfies ϕ under $\langle U, UR\rangle$ if and only if there are k sets in family F whose union is S.

If U is safe with respect to ϕ, by definition, a subset U' of U satisfies $\left(r_i \otimes \left(\bigotimes_{k-1} \mathsf{All}\right)\right)$ for every $i \in [1, n]$, which indicates users in U' together have membership of r_i for every $i \in [1, n]$. For any $i \in [1, n]$, U' satisfying $(r_i \otimes (\bigotimes_{k-1} \mathsf{All}))$ indicates that $|U'| = k$.

Suppose $U' = \{u_{a_1}, \ldots, u_{a_k}\}$. Because $(u_i, r_j) \in UR$ if and only if $e_j \in S_i$, the union of $\{S_{a_1}, \ldots, S_{a_k}\}$ is S. The answer to the SET COVERING problem is "yes".

On the other hand, without loss of generality, assume that $\bigcup_{i=1}^{k} S_i = S$. From the construction of UR, users $u_1, \ldots, u_k$ together have membership of r_i for every $i \in [1, n]$, which indicates that $\{u_1, \ldots, u_k\}$ satisfies ϕ_i for every $i \in [1, n]$. Hence, $\{u_1, \ldots, u_k\}$ satisfies ϕ and U is safe with respect to ϕ. ■

A.5 Proof of Theorem 2.5.2

Lemma A.5.1 $SSC\langle \neg, +, \sqcup, \sqcap, \odot, \otimes \rangle$ is in $\mathbf{coNP^{NP}}$.

Proof We show that the complement of $SSC\langle \neg, +, \sqcup, \sqcap, \odot, \otimes \rangle$ is in $\mathbf{NP^{NP}}$. Because SAFE is in $\mathbf{NP}$ (see Table 2.2), an $\mathbf{NP}$ oracle can decide whether a userset is safe with respect to a term. We construct a nondeterministic Oracle Turing Machine M that accepts an input consisting of a state $\langle U, UR, UP \rangle$ and a policy $\mathsf{sp}\langle P, \phi \rangle$ if and only if $\langle U, UR, UP \rangle$ is not safe with respect to $\mathsf{sp}\langle P, \phi \rangle$. M nondeterministically selects a set U of users in $\langle U, UR, UP \rangle$. If U does not cover P, then M rejects. Otherwise, M invokes the $\mathbf{NP}$ oracle to check whether U is safe with respect to ϕ. If the oracle answers "yes", then M rejects; otherwise, M accepts, as it has found a userset that covers P but is not safe with respect to ϕ, which violates the static safety policy. The construction of M shows that the complement of $SSC\langle \neg, +, \sqcup, \sqcap, \odot, \otimes \rangle$ is in $\mathbf{NP^{NP}}$. Hence, $SSC\langle \neg, +, \sqcup, \sqcap, \odot, \otimes \rangle$ is in $\mathbf{coNP^{NP}}$. ■

Lemma A.5.2 $SSC\langle \sqcup, \odot \rangle$ *is* $\mathbf{coNP}$*-hard.*

Proof We reduce the $\mathbf{coNP}$-complete VALIDITY problem for propositional logic to $SSC\langle \sqcup, \odot \rangle$. Given a propositional logic formula φ in disjunctive normal form, let $\{v_1, \ldots, v_n\}$ be the set of propositional variables in φ.

We create a state $\langle U, UR, UP \rangle$ with n permissions $p_1, p_2, \ldots, p_n$, $2n$ users $u_1, u'_1, u_2, u'_2, \ldots, u_n, u'_n$, and $2n$ roles $r_1, r'_1, r_2, r'_2, \ldots, r_n, r'_n$. We have $UP = \{(u_i, p_i), (u'_i, p_i) \mid 1 \leq i \leq n\}$ and $UR = \{(u_i, r_i), (u'_i, r'_i) \mid 1 \leq i \leq n\}$. We also construct a term ϕ from the

formula φ by replacing each literal v_i with r_i, each literal $\neg v_i$ with r'_i, each occurrence of $\wedge$ with $\odot$ and each occurrence of $\vee$ with $\sqcup$.

Note that X is safe with respect to $\phi_1 \sqcup \phi_2$ if and only if X is safe respect to either ϕ_1 or ϕ_2, and X is safe with respect to $\phi_1 \odot \phi_2$ if and only if X is safe respect to both ϕ_1 and ϕ_2. Thus the logical structure of ϕ follows that of φ.

We now show that the formula φ is valid if and only if $\langle U, UR, UP\rangle$ is safe with respect to the policy $\mathsf{sp}\langle\{p_1, p_2, \ldots, p_n\}, \phi\rangle$. On the one hand, if the formula φ is not valid, then there is an assignment I that makes it false. Using that assignment, we construct a userset $X = \{u_i \mid I(v_i) = \text{true}\} \cup \{u'_i \mid I(v_i) = \text{false}\}$. X covers all permissions in P, but X is not safe with respect to ϕ. On the other hand, if $\langle U, UR, UP\rangle$ is not safe with respect to $\mathsf{sp}\langle\{p_1, p_2, \ldots, p_n\}, \phi\rangle$, then there exists a set X of users that covers P but X is not safe with respect to ϕ. In order to cover all permissions in P, for each $i \in [1, n]$, at least one of u_i, u'_i is in X. Without loss of generality, assume that for each i, exactly one of u_i, u'_i is in X. (If both u_i, u'_i are in X, we can remove either one, the resulting set is a subset of X and still covers P.) Then we can derive a truth assignment I from X by setting v_i to true if $u_i \in X$ and to false if $u'_i \in X$. Then the formula evaluates to false, because X is not safe with respect to ϕ. ■

Lemma A.5.3 $SSC\langle\sqcap, \odot\rangle$ is **NP**-hard.

Proof There is a straightforward reduction from $SAFE\langle\sqcap, \odot\rangle$ to $SSC\langle\sqcap, \odot\rangle$. Given a term ϕ using only operators $\sqcap$ or $\odot$, in order to check whether a userset X is safe with respect to ϕ, we can construct a policy $\mathsf{sp}\langle P, \phi\rangle$ and a state $\langle U, UR, UP\rangle$ such that X is the only set of users in the state that covers P. In this case, X is safe with respect to ϕ if and only if the state we constructed satisfies $\mathsf{sp}\langle P, \phi\rangle$. Since $SAFE\langle\sqcap, \odot\rangle$ is **NP**-hard (see Table 2.2), $SSC\langle\sqcap, \odot\rangle$ is **NP**-hard. ■

Lemma A.5.4 $SSC\langle\otimes\rangle$ *is* **coNP**-*hard.*

Proof We can reduce the **NP**-complete SET COVERING problem to the complement of $SSC\langle\otimes\rangle$. In SET COVERING, we are given a family $F = \{S_1, \ldots, S_m\}$ of subsets of a

finite set $S = \{e_1, \ldots, e_n\}$ and an integer k, where k is an integer smaller than m and n. We are asking whether there is a subfamily of sets $F' \subseteq F$ whose union is S and $|F'| \leq k$.

Given an instance of the Set Covering problem, construct a state $\langle U, UR, UP\rangle$ such that $UR = \{(u_i, r_i) \mid i \in [1, m]\}$ and $UP = \{(u_i, p_j) \mid e_j \in S_i\}$. Construct a safety policy $\mathsf{sp}\langle P, \phi\rangle$, where $P = \{p_1, \ldots, p_n\}$ and $\phi = (\bigotimes_{k+1} \mathsf{All})$. ϕ is satisfied by any set of no less than $k + 1$ users.

First, if $\langle U, UR, UP\rangle$ is safe, no k users together have all permissions in P. In this case, since u_i corresponds to S_i, there does not exist k sets in family F whose union is S. The answer to the Set Covering problem is "no".

Second, if $\langle U, UR, UP\rangle$ is not safe, there exist a set of no more than k users together have all permissions in P. Accordingly, the answer to the Set Covering problem is "yes".

Since the SET COVERING problem is **NP**-complete, we conclude that the complement of $SSC\langle\otimes\rangle$ is **NP**-hard. Hence, $SSC\langle\otimes\rangle$ is **coNP**-hard. ■

Tractable cases of SSC:

Lemma A.5.5 $SSC\langle\neg, +, \sqcap, \sqcup\rangle$ is in **P**.

Proof Given a term ϕ with operators $\neg, +, \sqcap$ and $\sqcup$, construct another term ϕ' by removing $+$ in ϕ. For example, if $\phi = ((r_1 \sqcap r_2)^+ \sqcup r_3^+)$, then $\phi' = ((r_1 \sqcap r_2) \sqcup r_3)$. When only operators $\neg, +, \sqcap$ and $\sqcup$ are allowed, if a set U of users satisfies ϕ, then there exists $U' \subseteq U$ such that U' satisfies ϕ'. This indicates that U is safe with respect to ϕ if and only if U is safe with respect to ϕ'. Therefore, in order to show that $SSC\langle\neg, +, \sqcap, \sqcup\rangle$ is tractable, it suffices to prove that $SSC\langle\neg, \sqcap, \sqcup\rangle$ is in **P**.

A term ϕ' with operators $\neg, \sqcap$ and $\sqcup$ may be satisfied only by singleton. A state $\langle U, UR, UP\rangle$ is safe with respect to $\mathsf{sp}\langle\{p_1, \ldots, p_m\}, \phi'\rangle$, if and only if for any set U of users who together have all permissions in $\{p_1, \ldots, p_m\}$, there exists a user $u \in U$ such that $\{u\}$ satisfies ϕ'. This is equivalent to checking whether there exists a permission p_i ($i \in [1, m]$) such that for every user u having p_i, $\{u\}$ satisfies ϕ'. The following algorithm performs such a check.

```
isSafe(P, φ′, UR, UP)
begin
   For each p_i in {p_1,...,p_m} do
      flag = true;
      For each u such that (u, p_i) ∈ UP do
         If u does not satisfy φ′ then
            flag = false;
            break;
         EndIf;
      EndFor;
      If flag then return true;
   EndFor;
   return false;
end
```

The worst-case time complexity of the above algorithm is $O(m \times |U| \times t)$, where t is the time taken to check whether a singleton satisfies a term with operators $\neg, \sqcap$ and $\sqcup$, which is polynomial in the size of input according to Theorem 2.4.1. ■

Lemma A.5.6 *SSC* $\langle \neg, +, \odot \rangle$ *is in* **P**.

Proof The general form of terms built using only $\neg, +$ and $\odot$ is $(\gamma_1 \odot \cdots \odot \gamma_n)$, where γ_i is of the form r, $\neg r$, r^+ or $(\neg r)^+$, where r is a role. Given a term ϕ with operators $\neg, +$ and $\odot$, construct another term ϕ' by removing $+$ in ϕ. It is clear that if a set U of users satisfies ϕ, then there exists $U' \subseteq U$ such that U' satisfies ϕ'. This indicates that U is safe with respect to ϕ if and only if U is safe with respect to ϕ'. Therefore, in order to show that SSC $\langle \neg, +, \odot \rangle$ is tractable, it suffices to prove that SSC $\langle \neg, \odot \rangle$ is in **P**.

Given a policy $\mathsf{sp}\langle \{p_1, \ldots, p_m\} \rangle$, without loss of generality, assume that $\phi' = (\gamma_1 \odot \cdots \odot \gamma_n)$, where $\gamma_i = r$ or $\neg r$. The following algorithm checks whether $\langle U, UR, UP \rangle$ is safe with respect to ϕ'.

```
isSafe(P, φ', UR, UP)
begin
   Γ = {γ1, ... γn};
   For each pi in {p1,...,pm} do
      G_pi = ∅
      For each u such that (u, pi) ∈ UP do
         G_pi = G_pi ∪
         { γi ∈ φ' | u does not satisfy γi;}
      EndFor;
      Γ = Γ ∩ G_pi
   EndFor;
   if (Γ == ∅) return true
   else return false
end
```

In the above algorithm, G_{p_i} stores the set of sub-terms in ϕ' such that, for every $\gamma_j \in G_{p_i}$, there exists a user who has p_i but does not satisfy γ_j. At the end of the algorithm, on the one hand, if Γ contains a sub-term γ_i, it means that for every permissions p_j in $\{p_1, \ldots, p_n\}$, there exists a user u_{p_j} such that u_{p_j} has permission p_j but does not satisfy γ_i. In this case, the set of users $\{u_{p_1}, \ldots, u_{p_n}\}$ have all permissions in $\{p_1, \ldots, p_n\}$ but does not satisfy γ_i, and hence does not satisfy ϕ'. On the other hand, $\Gamma = \emptyset$ indicates that if users in U have all permissions in $\{p_1, \ldots, p_n\}$ then every sub-term γ_i in ϕ' is satisfied by a certain user in U. Therefore, there exists $U' \subseteq U$ such that U' satisfies ϕ'.

The worst-case time complexity of the above algorithm is $O(m \times |U| \times t)$, where t is the time taken to check whether a singleton satisfies a term with operators $\neg$ and $\odot$, which is polynomial according to Theorem 2.4.1. ■

Appendix B Proofs in Chapter 4

B.1 Proofs in Section 4.2

Proof to Lemma 4.2.3: WSP is **NP**-hard in R^{2}BAC, if the workflow uses constraints of the form $\langle \neq (s_1, s_2) \rangle$.

Proof To prove the problem is **NP**-hard, we reduce the **NP**-complete GRAPH K-COLORABILITY problem to this problem. In the GRAPH K-COLORABILITY problem, we are given a graph $G(V, E)$ and an integer k, and are asked whether we can assign no more than k colors to vertices in V such that every vertex has one color and vertices n_i and n_j have different colors whenever $(n_i, n_j) \in E$.

Given a graph $G(V, E)$, we construct a workflow $W = \langle S, \preceq, SA, C \rangle$ and an access control state $\gamma = \langle U, UR, B \rangle$ such that there is a one-to-one correspondence between steps in S and vertices in V. Let $U = \{u_1, \cdots, u_k\}$, where each $u_i \in U$ corresponds to a color. Construct UR and SA in such a way that every user in U is authorized to perform every step in S. For every $(n_i, n_j) \in E$, construct a constraint $\langle \neq (s_i, s_j) \rangle$, which requires that s_i and s_j must be performed by different users. If G is k-colorable, then we can construct a plan P such that s_j is performed by u_i if and only if n_j is assigned the ith color. Since no pair of adjacent vertices have the same color, no pair of steps restricted by a constraint is assigned to the same user in P. Hence, P satisfies all constraints and is a valid plan. Similarly, if there is a valid plan P for W in γ, we can find a way to color G with no more than k colors based on plan P. In general, G is k-colorable if and only if W is satisfiable. ■

Proof to Lemma 4.2.4: WSP is **NP**-hard in R^{2}BAC, if the workflow uses constraints of the form $\langle = (s, \exists X) \rangle$.

Proof To prove the problem is **NP**-hard, we reduce the **NP**-complete HITTING SET problem to this problem. In the HITTING SET problem, we are given a set Z and a family $F = \{Z_1, \cdots, Z_m\}$ of subsets of Z and are asked whether there exists a size-k subset H of Z such that, for every $Z_i \in F$, $H \cap Z_i \neq \emptyset$.

We construct a workflow $W = \langle S \cup A, \preceq, SA, C\rangle$ and an access control state $\gamma = \langle U, UR, B\rangle$ such that the answer to the HITTING SET problem is "yes" if and only if W is satisfiable under γ. Let $U = \{u_i \mid e_i \in Z\}$ be a set of users. Let $S = \{s_1, \cdots, s_k\}$ be a set of k steps. Construct UR and SA in such a way that every step in S is authorized to all users in U. Furthermore, let $A = \{a_1, \cdots, a_m\}$ be a set of m steps and $S \cap A = \emptyset$. Construct UR and SA in such a way that u_i is authorized to perform a_j if and only if $e_i \in Z_j$. Intuitively, S corresponds to H and each step $a_i \in A$ corresponds to $Z_i \in F$. Finally, construct a set $C = \{c_1, \cdots, c_m\}$ of m constraints, where $c_i = \langle = (a_i, \exists S)\rangle$.

On the one hand, assume that P is a valid plan. Let $H = \{e_i \mid \exists_{s_j}(u_i, s_j) \in P\}$. For every $a_i \in A$, let u_j be the user such that $(u_j, a_i) \in P$. P being valid indicates that u_j is authorized to perform a_i. From our construction, we have $e_j \in Z_i$. Furthermore, for every $i \in [1, m]$, $\langle = (a_i, \exists S)\rangle$ being satisfied indicates that there exists $s_l \in S$ such that $(u_j, s_l) \in P$. And $(u_j, s_l) \in P$ indicates that $e_j \in H$. Therefore, we have $H \cap Z_i = e_j$. In general, for every $Z_i \in F$, $H \cap Z_i \neq \emptyset$. The answer to the HITTING SET problem is "yes".

On the other hand, assume that the answer to the HITTING SET problem is "yes". We now construct a plan P that satisfies the workflow. Without loss of generality, assume that $H = \{e_1, \cdots, e_k\}$. We initialize P to $\emptyset$ and add (u_i, s_i) to P for every $i \in [1, k]$. Recall that s_i is authorized to every user in U. For every $Z_j \in F$, add (u_i, a_j) to P when $H \cap Z_j = e_i$. $e_i \in Z_j$ implies that u_i is authorized to perform a_j. Furthermore, for every $c_j \in C$ (remind that $c_j = \langle = (a_j, \exists S)\rangle$), $(u_i, a_j) \in P$ and $(u_i, s_i) \in P$ indicate that c_j is satisfied. Therefore, P is a valid plan. ■

Proof to Theorem 4.3.2: WSP is in **FPT** in R^{2}BAC, if = and $\neq$ are the only binary relations used by constraints in the workflow.

Proof Given a workflow $W = \langle S, \preceq, SA, C\rangle$ and an access control state $\gamma = \langle U, UR, B\rangle$, let k be the number of steps in W. The description of the algorithm is as follow.

1. For every step $s_i \in S$, compute the set $AU(s_i)$ of users who are authorized to perform s_i according to UR and SA.

2. Process every constraint c in the form of $\langle = (s_1, s_2)\rangle$. Let $U' = AU(s_1) \cap AU(s_2)$ be the set of users who are authorized to perform both s_1 and s_2. If $U' = \emptyset$, then c is not satisfiable and neither nor W. Otherwise, we set $AU(s_1)$ and $AU(s_2)$ to be U'.

3. Let C' be the set of all constraints in C that are in the form of $\langle = (s, \exists X)\rangle$. We construct a search tree as follow.

   Label the root of the tree with C' and $UA = \{AU(s_i) \mid s_i \in S\}$. Choose a constraint c from C'. Without loss of generality, assume that $c = \langle = (s_0, \exists\{s_1, \cdots, s_m\})\rangle$ ($m \leq k-1$). s_0 must be performed by the same user as s_1, or s_2, $\cdots$, or s_m. We create m children of the root corresponding to these m possibilities. Let $U_{0,1} = AU(s_0) \cap AU(s_1)$. If $U_{0,1} = \emptyset$, then the first child of the root is marked as "invalid" and will not be further processed, because it is impossible to find a user who is authorized to perform both s_0 and s_1. Otherwise, the first child is labeled with $C' - \{c\}$ and UA_1, where UA_1 is the same as UA except that $AU(s_0)$ and $AU(s_1)$ are set to be $U_{0,1}$. Intuitively, the set of constraints labeling a node represents the remaining constraints, while the set UA labeling a node represents the user-step authorization that satisfies those constraints that have been processed. The other $m-1$ children of the root are processed similarly. We then recursively process the children of the children of the root and so on until all nodes in the tree have been processed. We then say that the search tree is fully-developed.

   In a fully-developed search tree, a leave node that is not marked as "invalid" (in this case, it must have been labeled with an empty set of constraints) is called "alive". If there is no "alive" leave node in the search tree, then it is impossible to satisfy all constraints in C' and thus W is not satisfiable.

   Note that there are no more than $k2^{k-1}$ different constraints in the form of $\langle = (s, \exists X)\rangle$ as s and X can take at most k and 2^{k-1} different values, respectively [1]. Therefore,

[1] The 2^{k-1} upper-bound is loose and can be improved, but it suffices to prove the result we want.

the depth of the fully-developed search tree is no more than $k2^{k-1}$. Furthermore, the number of children of each node is bounded by $k-1$. Hence, the size of the fully-developed search tree is bounded by $(k-1)^{k2^{k-1}}$. Processing each node in the search tree involves computing no more than $k-1$ intersections and can be done in $O(kn)$.

4. For each "alive" leave node v in the search tree, we check whether all constraints using $\neq$ can be satisfied with the user-step authorization UA labeling v. According to Lemma 4.3.1, this can be done in $O(k^{k+1}n)$, where n is the size of the entire input to the problem. If the answer is "yes" for any "alive" leave node, the workflow W is satisfiable; otherwise, W is not satisfiable.

In general, the above algorithm finishes in $O(f(k)n)$ where $f(k) = k^{k+1}(k-1)^{k2^{k-1}}$. Hence, the problem is in $\mathbf{FPT}$. ■

Proof to Theorem 4.3.3: WSP is $W[1]$-hard in R^{2}BAC if user-defined binary relations are used in constraints.

Proof We show that WSP is $W[1]$-hard even if the workflow only has constraints in the form of $\langle\rho(s_1, s_2)\rangle$, where ρ is a user-defined binary relation. Because $\langle\rho(s_1, s_2)\rangle$ can be equivalently represented as $\langle\rho(s_1, \exists\{s_2\})\rangle$, the problem is $W[1]$-hard even if the workflow only has constraints in the form of $\langle\rho(s, \exists X)\rangle$.

We reduce INDEPENDENT SET to WSP. In INDEPENDENT SET, we need to determine whether there is a size-k independent set in graph $G(V, E)$. An independent set of G is a set of vertices V' such that $V' \subseteq V$ and no pair of vertices in V' are adjacent to each other in G. INDEPENDENT SET with parameter k is $W[1]$-complete.

Given an integer k and a graph $G(V, E)$ where $V = \{v_1, \cdots, v_m\}$, we construct a workflow $W = \langle S, \preceq, SA, C\rangle$ and an access control state $\gamma = \langle U, UR, B\rangle$, where $U = \{u_1, \cdots, u_m\}$. There is a one-to-one correspondence between users in U and vertices in V. $S = \{s_1, \cdots, s_k\}$ and $s_1 \preceq \cdots \preceq s_k$. Let $UR = \{(u_i, r) \mid i \in [1, m]\}$ and $SA = \{(r, s_i) \mid i \in [1, k]\}$. In other words, every user in U is authorized to perform every step in S. B contains one binary relation ρ, and $\rho = \{(u_i, u_j) \mid i \neq j \wedge (v_i, v_j) \notin E\}$. Intuitively, $(u_i, u_j) \in \rho$ if and only if $u_i \neq u_j$ and the vertices corresponding to the two

users are not adjacent to each other in G. For every $i \in [2, k]$, we construct $i-1$ constraints $c_{i,1}, \cdots, c_{i,i-1}$ such that $c_{i,j} = \langle \rho(s_i, s_j) \rangle$ where $j \in [1, i-1]$.

Next, we show that G has a size-k independent set if and only if W is satisfiable under γ.

On the one hand, without loss of generality, assume that $\{v_1, \cdots, v_k\}$ is an independent set of G. By definition of independent set, we have $(v_i, v_j) \notin E$, for any $i, j \in [1, k]$ and $i \neq j$. We construct a plan $P = \{(u_i, s_i) \mid i \in [1, k]\}$. For any $i, j \in [1, k]$ and $i \neq j$, $(v_i, v_j) \notin E$ implies that $(u_i, u_j) \in \rho$. Therefore, no constraint is violated by P and P is a valid plan.

On the other hand, assume that there is a valid plan P for W. From the construction of ρ, the k steps in W must be performed by k different users. Without loss of generality, assume that $P = \{(u_i, s_i) \mid i \in [1, k]\}$. Since no constraint is violated by P, we have $(u_i, u_j) \in \rho$ for any $i, j \in [1, k]$ and $i < j$. Let $V' = \{v_1, \cdots, v_k\}$. For any pair of vertices (v_i, v_j) where $i, j \in [1, k]$ and $i < j$, $(u_i, u_j) \in \rho$ implies that $(v_i, v_j) \notin E$. Hence, V' is a size-k independent set of G.

Finally, we show that the above reduction is a fixed-parameter reduction. In our reduction, the parameter k of the INDEPENDENT SET instance has the same value as the number of steps in the corresponding WSP instance. Furthermore, the number m of users in the workflow is the same as the number of vertices in the graph. ρ can be generated in quadratic time to the size of G. There are no more than $k^2/2$ constraints in the workflow, and UR contains m items while SA contains k items. In general, the WSP instance can be generated from the INDEPENDENT SET instance in $O(n^2 + k^2)$, where n is the size of graph G. Hence, the reduction is a fixed-parameter reduction. ■

Proof to Theorem 4.3.4: WSP in R^{2}BAC is in $W[2]$.

Proof We reduce WSP to the weighted satisfiability problem of decision circuits of weft 2 (denoted as $WCS[2]$). In the following, we encode an WSP instance into a boolean expression that can be represented as a decision circuit of weft 2. And the answer to the WSP instance is "yes" if and only if the answer to the $WCS[2]$ instance is "yes".

Given a workflow $W = \langle S, \preceq, SA, C\rangle$ and an access control state $\gamma = \langle U, UR, B\rangle$, let $S = \{s_1, \cdots, s_k\}$ and $U = \{u_1, \cdots, u_n\}$. We construct kn variables $v_{i,j}$ where $i \in [1, k]$ and $j \in [1, n]$. Intuitively, setting $v_{i,j}$ to true corresponds to assigning user u_j to s_i.

Let $AU(s)$ be the set of authorized users for step s. For every $s_i \in S$, we construct a clause $H_{s_i} = \bigvee_{u_j \in AU(s_i)} v_{i,j}$, which indicates that s_i must be performed by an authorized user. The length of such a clause is no more than n and there are k such clauses. Note that a weight-k truth assignment that satisfy all the k clauses (i.e. $H_{s_1}, \cdots, H_{s_k}$) must set exactly one $v_{i,j}$ to true for every $i \in [1, k]$, which indicates that every step is assigned to exactly one user.

For every constraint $c \in C$, we construct clauses for c as follows. Given a set $F = \{f_1, \cdots, f_m\}$ of clauses, we define $\bigvee F$ as $f_1 \vee \cdots \vee f_m$ and $\bigwedge F$ as $f_1 \wedge \cdots \wedge f_m$.

- When $c = \langle \rho(s_{i_1}, s_{i_2})\rangle$: Let $F = \{v_{i_1,j_1} \wedge v_{i_2,j_2} \mid u_{j_1} \in AU(s_{i_1}) \wedge u_{j_2} \in AU(s_{i_2}) \wedge (u_{j_1}, u_{j_2}) \in \rho\}$. We construct a clause $H_c = \bigvee F$, which indicates that s_{i_1} and s_{i_2} must be performed by a pair of authorized users that satisfies ρ.
- When $c = \langle \rho(s, \exists X)\rangle$: Without loss of generality, assume that $c = \langle \rho(s_0, \exists\{s_1, \cdots, s_m\})\rangle$. For every $i \in [1, m]$, let $F_i = \{v_{0,j_1} \wedge v_{i,j_2} \mid u_{j_1} \in AU(s_0) \wedge u_{j_2} \in AU(s_i) \wedge (u_{j_1}, u_{j_2}) \in \rho\}$. We construct a clause $H_c = \bigvee F_1 \vee \cdots \vee \bigvee F_m$, where $\bigvee F_i$ indicates that s_0 and s_i must be performed by a pair of authorized users that satisfies ρ.

Let $F = \{H_{s_i} \mid s_i \in S\} \cup \{H_c \mid c \in C\}$. $H = \bigwedge F$ is a clause encoding the WSP instance in the sense that H has a weight-k satisfying truth assignment if and only if W is satisfiable under γ. H can be represented by a decision circuit using a large "$\wedge$" gate that connects a number of large "$\vee$" gates that connect either a number of variables or a number of small "$\wedge$" gates, each of which connects two variables. The decision circuit is thus a weft 2 decision circuit.

In the above reduction, the number of step in the WSP instance is the same as the weight k of the corresponding $WCS[2]$ instance. There are k H_s clauses, each of which has length no more than n, where n is the size of the WSP instance. And there are n H_c clauses, each

of which has length no more than kn^2. Hence, the construction of the decision circuit can be done in $O(kn^3)$. Therefore, the above reduction is a fixed-parameter reduction and WSP is in $W[2]$. ■

B.2 Proofs in Section 4.4.1

Proof to Theorem 4.4.3: CRCP is **PSPACE**-complete.

Proof The two-person game of decremental resiliency has the following two properties, which indicates that it can be solved in **PSPACE**.

1. The number of rounds is bounded by a polynomial in the size of the input. In particular, the game must come to a conclusion after at most k rounds, where k is the number of steps in the workflow.
2. Given an intermediate state, which consists of a partial plan, the set of remaining users and the set of unfinished steps, there is a polynomial-space algorithm that constructs all possible combinations of actions of the two users in the next round, and determines if the game is over.

To show PSPACE-hardness, we reduce the **PSPACE**-complete QUANTIFIED SATISFIABILITY (or QSAT) problem to CRCP. In the QSAT, we are given a boolean expression ϕ in conjunction normal form (CNF), with boolean variables $x_1, \cdots, x_m$. Is it true that there is a truth value for x_1 such that for both truth value of x_2 there exists a truth value for x_3, and so on up to x_m, ϕ is satisfied by the overall truth assignment? In other words,

$$\exists_{x_1} \forall_{x_2} \exists_{x_3} \cdots Q_{x_m} \phi?$$

where Q is "exists" if m is odd, or "for all" if m is even. Without loss of generality, we assume that m is odd.

The QSAT problem can be modeled as a two-person game, in which Player 1 and Player 2 control the truth assignment of variables in $\{x_i \mid i \in [1, m] \wedge i \text{ is odd}\}$ and $\{x_j \mid j \in [1, m] \wedge j \text{ is even}\}$, respectively. Player 1 tries to satisfy ϕ, while Player 2 tries to prevent this.

Given a QSAT instance $\exists_{x_1}\forall_{x_2}\exists_{x_3}\cdots\exists_{x_m}\phi$ where $\phi = \phi_1 \wedge \cdots \wedge \phi_k$, we construct a CRCP instance. The detailed construction of the CRCP instance is given in Figure B.1.

Next, we prove that the answer to the CRCP instance is "yes" if and only if the answer to the QSAT instance is "yes". In the constructed workflow $W = \langle S, \preceq, SA, C\rangle$, S consists of three parts A, B and D. Steps in A determine truth values of variables $x_1, \cdots, x_m$. Intuitively, assigning user u_i (or v_i) to a_i represents setting x_i to "true" (or "false"). Steps in B correspond to the k clauses in ϕ. Steps in D are used to restrict the behaviors of the two players.

We need to to prove the following four claims.

1. For every even number $i \in [1, m]$, Player 2 should remove either u_i or v_i right after the execution of a_{i-1}. In other words, Player 2 controls the user-step assignment for steps in $\{a_i \mid a_i \in A \wedge i \text{ is even}\}$.
2. For every step $a_i \in A$, Player 1 should assign either u_i or v_i to a_i, when Player 2 plays optimally.
3. If Player 1 plays optimally, then steps in D can always be completed.
4. If both players play optimally, all steps in B can be completed if and only if the truth assignment of boolean variables $x_1, \cdots, x_m$ corresponding to the user-step assignment of steps in A satisfies ϕ.

If Claim 1 and Claim 2 are true, then Player 1 and Player 2 control the truth assignment of variables in $\{x_i \mid i \in [1, m] \wedge i \text{ is odd}\}$ and $\{x_j \mid j \in [1, m] \wedge j \text{ is even}\}$, respectively. If Claim 3 and Claim 4 are true, then the workflow instance can be completed if and only if ϕ is satisfied by the truth assignment. In general, the answer to the CRCP instance is "yes" if and only if the answer to the QSAT instance is "yes".

The proofs to the four claims are listed as follows.

Proof to Claim 1: First of all, since the total number of absent users is bounded by t, Player 2 should not remove any of those users with $t + 1$ copies. Users in $\{u_i, v_i \mid i \in [1, m] \wedge i \text{ is even}\}$ are unique and are the only users with less than than $t + 1$ copies.

Secondly, given an even number i, if Player 2 removes both u_i and v_i, then there must exist an even number $j \in [1, m]$ such that both u_j and v_j are available throughout the game, as Player 2 can remove at most $(m-1)/2$ users. In this case, Player 1 can assign u' to all remaining even steps in A as well as all steps in B, and then assign u_j to d_1 and v_j to d_2. Such an assignment complete the workflow without violating any constraint. Player 1 wins. Therefore, Player 2 should remove either u_i or v_i for every even number i.

Finally, we would like to point out that Player 2 should remove u_i or v_i before the execution of a_i, where i is a even number. If Player 2 does this after the execution of a_i, then Player 1 gains advantage by being able to choose between u_i and v_i for a_i. However, removing u_i or v_i after a_i does not affect future user-step assignment, as it is p_i and q_i rather than u_i and v_i that will be performing steps in B.

Proof to Claim 2: The statement is true when i is odd, since u_i and v_i are the only users authorized to perform a_i. In the following, we only discuss the case when i is even.

From Claim 1, when Player 2 plays optimally, he/she removes either u_i or v_i for every even number $i \in [1, m]$. Given an even number i, without loss of generality, assume that Player 2 removes u_i. In this case, Player 1 may either assign v_i or u' to a_i. If, by contradiction, Player 1 assigns u' to a_i, then according to constraint $\langle \overline{\rho_1}(d_1, \forall A_{even}) \rangle$, Player 1 cannot assign v' to d_1 as $(v', u') \in \rho_1$. Thus, Player 1 has to choose a certain u_j or v_j for d_1, where j is even. By the time d_1 is to be executed, either u_j or v_j must have been removed by Player 2. Without loss of generality, assume that u_j is available and is thus assigned to d_1. According to $\langle \rho_2(d_2, d_1) \rangle$, Player 1 has to assign v_j to d_2, but v_j is not available. Hence, d_2 cannot be completed and Player 1 losses. Therefore, Player 1 must not assign u' to a_i when Player 2 plays optimally. The only choice for Player 1 is to assign v_i to a_i.

Proof to Claim 3: We have shown that if Player 2 does not follow the strategy in Claim 1, then Player 1 can complete all steps in the workflow. When both players play optimally, according to Claim 1 and Claim 2, only users in $\{u_i, v_i \mid i \in [1, m]\}$ are assigned to steps in A. In this case, Player 1 can assign v' to d_1 and u' to d_2 without violating any constraints.

Proof to Claim 4: From Claim 1 and Claim 2, when both players play optimally, only users in $\{u_i, v_i \mid i \in [1, m]\}$ are assigned to steps in A. For any $b_j \in B$, according to

constraint $\langle \rho_0(b_j, \exists A) \rangle$, u' cannot be assigned to b_j, as $(u', u_i), (u', v_i) \notin \rho_0$. Furthermore, p_i and q_i correspond to u_i and v_i respectively according to ρ_0. From the construction of SA and UR, p_i (or q_i) is authorized to perform b_j if and only if setting x_i to true (or false) satisfies clause ϕ_j. Hence, Player 1 can assign a user to b_j if and only if the truth assignment determined by the user-step assignment of steps in A satisfies ϕ_j. In general, all steps in B can be completed if and only if ϕ_j is satisfied for every $j \in [1, k]$, which indicates that ϕ is satisfied. ■

Proof to Theorem 4.4.4: DRCP is $\mathbf{PSPACE}$-complete.

Proof The proof that DRCP is in $\mathbf{PSPACE}$ is similar to the case of CRCP. In the following, we only prove that the problem is $\mathbf{PSPACE}$-hard.

We reduce the $\mathbf{PSPACE}$-complete QUANTIFIED SATISFIABILITY (or QSAT) problem to DRCP. Given a QSAT instance $\exists_{x_1} \forall_{x_2} \exists_{x_3} \cdots \exists_{x_m} \phi$ where $\phi = \phi_1 \wedge \cdots \wedge \phi_k$, we construct a DRCP instance. The detailed construction of the DRCP instance is given in Figure B.2.

We need to prove that the answer to the DRCP instance is "yes" if and only if the answer to the QSAT instance is "yes". In the constructed workflow $W = \langle S, \preceq, SA, C \rangle$, S consists of two parts A and B. Steps in A determine truth values of variables $x_1, \cdots, x_m$. Intuitively, assigning user u_i (or v_i) to a_i represents setting x_i to "true" (or "false"). Steps in B correspond to the k clauses in ϕ.

First of all, it is clear that Player 2 should remove one user in each round. However, since there are two copies of u_i and v_i for odd number $i \in [1, m]$, and two copies of p_j and q_j for $j \in [1, k]$, Player 2's action only affects the user-step assignment of a_i for even number $i \in [1, m]$. Therefore, Player 1 and Player 2 has control over the user-step assignment of odd number steps in A and even number steps in A, respectively. A user-step assignment for steps in A represents a truth assignment for variables $x_1, \cdots, x_m$.

Secondly, according to relation ρ, p_i and q_i correspond to u_i and v_i respectively. According to the construction of SA and UR, p_i (or q_i) is authorized to perform b_j if and only if setting x_i to true (or false) satisfies clause ϕ_j. Due to the constraint $\langle \rho(b_j, \exists A) \rangle$, Player 1

can assign a user to b_j if and only if the truth assignment determined by user-step assignment in A satisfies ϕ_j. Therefore, Player 1 can complete all steps in B if and only if the truth assignment satisfies ϕ.

In general, Player 1 can always win the game if and only if the answer to the QSAT instance is "yes". ■

Input: $\exists_{x_1}\forall_{x_2}\exists_{x_3}\cdots\exists_{x_m}\phi$, where $\phi = \phi_1 \wedge \cdots \wedge \phi_k$

Output: A workflow $W = \langle S, \preceq, SA, C\rangle$, an integer $t = (m-1)/2$, an access control state $\gamma = \langle U, UR, \{\rho_0, \rho_1, \rho_2\}\rangle$

Construction of W and γ:

- **Steps and Step-Authorization:**

  $S = A \cup B \cup D$

  We have $A = \{a_1, \cdots, a_m\}$, $B = \{b_1, \cdots, b_k\}$, $D = \{d_1, d_2\}$, and $a_1 \preceq \cdots \preceq a_m \preceq d_1 \preceq d_2 \preceq b_1 \preceq \cdots \preceq b_k$

  $SA = \{(r_{a_i}, a_i) \mid a_i \in A\} \cup \{(r_{b_i}, b_i) \mid b_i \in B\} \cup \{(r_{d_1}, d_1), (r_{d_2}, d_2)\}$

- **Configuration:**

  $U = \{u_i, v_i, p_i, q_i \mid i \in [1, m]\} \cup \{u', v'\}$

  For every odd number i in $[1, m]$, there are $t+1$ copies of u_i and v_i. For every $j \in [1, m]$, there are $t+1$ copies of p_j and q_j. There are $t+1$ copies of u' and v' as well.

$$\begin{aligned} UR = &\{(u_i, r_{a_i}), (v_i, r_{a_i}) \mid i \in [1, m] \wedge i \text{ is odd}\} \\ &\cup \{(u_i, r_{a_i}), (v_i, r_{a_i}), (u', r_{a_i}) \mid i \in [1, m] \wedge i \text{ is even}\} \\ &\cup \{(u', r_{b_i}) \mid i \in [1, m]\} \cup \Upsilon_1 \cup \cdots \cup \Upsilon_k \\ &\cup \{(v', r_{d_1})\} \cup \{(u_i, r_{d_1}), (v_i, r_{d_1}) \mid i \in [1, m] \wedge i \text{ is even}\} \\ &\cup \{(u', r_{d_2})\} \cup \{(u_i, r_{d_2}), (v_i, r_{d_2}) \mid i \in [1, m] \wedge i \text{ is even}\} \end{aligned}$$

  Construction of Υ_i: Let L_i be the set of literals in clause ϕ_i. $(p_j, r_{b_i}) \in \Upsilon_i$ if and only if there exists a literal $l \in L_i$ such that $l = x_j$; and $(q_j, r_{b_i}) \in \Upsilon_i$ if and only if there exists a literal $l \in L_i$ such that $l = \neg x_j$.

- **Constraints:**

  $C = \{\langle \rho_0(b_i, \exists A)\rangle \mid i \in [1, k]\} \cup \{\langle \overline{\rho_1}(d_1, \forall A_{even})\rangle, \langle \rho_2(d_2, d_1)\rangle\}$

  where $A_{even} = \{a_i \mid i \text{ is even}\}$. We have

  - $\rho_0 = \{(p_i, u_i), (q_i, v_i) \mid i \in [1, m]\} \cup \{(u', u')\}$
  - $\rho_1 = \{(v', u')\}$
  - $\rho_2 = \{(u_i, v_i), (v_i, u_i) \mid i \text{ is even}\} \cup \{(u', v')\}$.

Figure B.1. Generating a CRCP instance for a QSAT instance.

Input:

$\exists_{x_1}\forall_{x_2}\exists_{x_3}\cdots\exists_{x_m}\phi$, where $\phi = \phi_1 \wedge \cdots \wedge \phi_k$

Output:

A workflow $W = \langle S, \preceq, SA, C\rangle$, an integer $t = 1$, an access control state $\gamma = \langle U, UR, \{\rho\}\rangle$

Construction of W and γ:

- **Steps and Step-Authorization:**

  $S = A \cup B$

  We have $A = \{a_1, \cdots, a_m\}$, $B = \{b_1, \cdots, b_k\}$, and $a_1 \preceq \cdots \preceq a_m \preceq b_1 \preceq \cdots \preceq b_k$

  $SA = \{(r_{a_i}, a_i) \mid a_i \in A\} \cup \{(r_{b_i}, b_i) \mid b_i \in B\}$

- **Configuration:**

  $U = \{u_i, v_i, p_i, q_i \mid i \in [1, m]\}$

  For every odd number i in $[1, m]$, there are 2 copies of u_i and v_i. For every $j \in [1, m]$, there are 2 copies of p_j and q_j.

  $$UR = \{(u_i, r_{a_i}), (v_i, r_{a_i}) \mid i \in [1, m]\} \cup \Upsilon_1 \cup \cdots \cup \Upsilon_k$$

  Construction of Υ_i: Let L_i be the set of literals in clause ϕ_i. $(p_j, r_{b_i}) \in \Upsilon_i$ if and only if there exists a literal $l \in L_i$ such that $l = x_j$; and $(q_j, r_{b_i}) \in \Upsilon_i$ if and only if there exists a literal $l \in L_i$ such that $l = \neg x_j$.

- **Constraints:**

  $C = \{\langle \rho(b_i, \exists A)\rangle \mid i \in [1, k]\}$
  where $\rho = \{(p_i, u_i), (q_i, v_i) \mid i \in [1, m]\}$

Figure B.2. Generating a DRCP instance for a QSAT instance.

VITA

VITA

Qihua Wang was born in Guangzhou, China. He attended the University of Science and Technology of China (USTC) in Hefei, China and obtained his bachelor's degree in computer science in July 2004. After he graduated from USTC, he enrolled in the Department of Computer Science at Purdue University where he obtained his master's degree in May 2007. He then spent seven months as a graduate-level co-op at the IBM Watson Research Center in New York from May 2007 to December 2007. In the summer of 2008, he worked as a graduate-level co-op at the IBM Almaden Research Center in California. He was a recipient of the Bilsland Dissertation Fellowship in 2009. He received the degree of Doctor of Philosophy in May 2009 under the direction of Professor Ninghui Li. His research interests are in computer security with a focus on access control, and in user-centered systems with a focus on computer-supported user collaboration.