On Mutually-Exclusive Roles and Separation of Duty

NINGHUI LI, MAHESH V. TRIPUNITARA, and ZIAD BIZRI
Purdue University

Separation of Duty (SoD) is widely considered to be a fundamental principle in computer security. A Static SoD (SSoD) policy states that in order to have all permissions necessary to complete a sensitive task, the cooperation of at least a certain number of users is required. Role-Based Access Control (RBAC) is today's dominant access control model. It is widely believed that one of RBAC's main strengths is that it enables the use of constraints to support policies such as Separation of Duty. In RBAC Statically Mutually Exclusive Roles (SMER) constraints are used to enforce SSoD policies. In this paper, we formulate and study fundamental computational problems related to the use of SMER constraints to enforce SSoD policies. We show that directly enforcing SSoD policies is intractable ($\mathbf{coNP}$-complete), while checking whether an RBAC state satisfies a set of SMER constraints is efficient; however, verifying whether a given set of SMER constraints enforces an SSoD policy is also intractable ($\mathbf{coNP}$-complete). We discuss the implications of these results. We show also how to generate SMER constraints that are as accurate as possible for enforcing an SSoD policy.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Complexity of proof procedures*

General Terms: Security, Theory

Additional Key Words and Phrases: role-based access control, separation of duty, constraints, verification, computational complexity

1. INTRODUCTION

Separation of Duty (SoD) is widely considered to be a fundamental principle in computer security [Clark and Wilson 1987; Saltzer and Schroeder 1975]. In its simplest form, the principle states that if a sensitive task is comprised of two steps, then a different user should perform each step. More generally, when a sensitive task is comprised of n steps, an SoD policy requires the cooperation of at least k (for some $k \leq n$) different users to complete the task. While the concept of SoD has long existed in the physical world, sometimes under the name "the two-man rule", in the computer security literature, its first mention seems to be by Saltzer and Schroeder [Saltzer and Schroeder 1975] under the name "separation of privilege." In Clark and Wilson's highly influential work [Clark and Wilson 1987] on

Authors' address: Department of Computer Sciences, Purdue University, 250 N. University Street, West Lafayette, IN 47907-2066, USA; email: {ninghui,mtripuni,zelbizri}@cs.purdue.edu.
A preliminary version of this paper appears in the *Proceedings of 2004 ACM Conference on Computer and Communications Security (CCS'04)* [Li et al. 2004].

commercial security policies for integrity protection, SoD was identified together with Well-formed Transformations as the two high-level mechanisms that are "at the heart of fraud and error control" and can be used to "enforce commercial security policies related to data integrity".

Consider the following example of buying and paying for goods, taken from the work by Clark and Wilson [Clark and Wilson 1987]. The steps that comprise such a task are: (1) ordering the goods and recording the details of the order; (2) recording the arrival of the invoice and verifying that the details on the invoice match the details of the order; (3) verifying that the goods have been received, and the features of the goods match the details on the invoice; and (4) authorizing payment to the supplier against the invoice. We would want to ensure that payment is not released on an order that was never placed, and that the received goods match those in the order and those in the invoice. A policy that requires a different user to perform each step may be too restrictive. It may be permissible, for instance, that the user who places the order also records the arrival of the invoice. One may require that (a) the cooperation of at least three users is needed to perform all four steps, and (b) two different users perform steps (1) and (4) (i.e., no single user can order goods and authorize payment for them).

There are at least two approaches to enforce an SoD policy. In one approach, the system maintains a history of each task instance (e.g., a particular order in the above example). The history includes information on who performed each step. Before a user performs a step on the instance, the system checks to ensure that the SoD policy is not violated. This is referred to as Dynamic SoD in the literature [Foley 1997; Nash and Poland 1990; Sandhu 1988]. (Foley [Foley 1997] refers to this as dynamic segregation of duties. Nash and Poland [Nash and Poland 1990] refer to this as object SoD and consider it as a kind of Dynamic SoD.) In Dynamic SoD, a user may be able to perform any step in a task instance; however, once the user has performed one step, the user cannot perform other steps in that instance.

Another approach to enforce SoD policies is to use Static SoD (SSoD) policies. Each SSoD policy states that no $k-1$ users together have all permissions to complete a sensitive task. Such an SSoD policy can be enforced by carefully assigning permissions to users, without maintaining a history for every task instance. It may seem that if an SSoD policy is satisfied, then the corresponding SoD policy is also satisfied. However, care must be taken to ensure this. Consider the example described above. Suppose that initially a user Bob has the permission to order goods. After placing an order, Bob's order permission is revoked and then Bob is assigned the permission to authorize payments. Now Bob can authorize a payment against the order he placed earlier. The SoD policy is violated even though Bob never has the order permission and payment permission at the same time. Such situations can be avoided, for example, by requiring that when a permission is revoked from a user, all active task instances that involve this permission are removed from the system first, or by treating such task instances specially (e.g., by maintaining a history for them).

Separation of Duty has been studied extensively in Role-Based Access Control (RBAC). Initially studied in database security research [Baldwin 1990; Ting 1988] about 15 years ago, RBAC [ANSI 2004; Ferraiolo et al. 1995; Ferraiolo and Kuhn 1992; Ferraiolo et al. 2003; Ferraiolo et al. 2001; Sandhu et al. 1996] has become today's dominant access control model. Over the past decade, interest in RBAC has increased dramatically, with most major information technology vendors offering products that incorporate some form of

RBAC in them. Today, all major database management systems support RBAC. In Windows Server 2003, Microsoft introduced Authorization Manager, which brings RBAC to the Windows family of operating systems. The commercial success of RBAC is often attributed to the following two advantages it has over previous approaches: reduced cost of administration and the ability to configure and enforce commercial security policies.

It has been recognized that "one of RBAC's great advantages is that SoD rules can be implemented in a natural and efficient way" [Ferraiolo et al. 2003]. In this paper we formulate and study fundamental computational problems related to the implementation of SoD in RBAC. In RBAC, permissions are associated with roles, and users are granted membership in appropriate roles, thereby acquiring the roles' permissions. RBAC uses mutual exclusion constraints to implement SoD policies.

The most common kind of constraints used limits the role membership of users. For example, one may declare two roles to be mutually exclusive in that no user is allowed to be a member of both roles. More generally, a constraint may require that no user is a member of t or more roles in a set of m roles $\{r_1, r_2, \cdots, r_m\}$. We call such constraints Statically Mutually Exclusive Roles (SMER) constraints. SMER constraints are part of most RBAC models, including the RBAC96 models by Sandhu et al. [Sandhu et al. 1996] and the proposed and adopted ANSI/NIST standard for RBAC [ANSI 2004; Ferraiolo et al. 2001]. Literature in RBAC also studies constraints that prevent any user from activating mutually exclusively roles simultaneously in a session. We call these constraints dynamic mutually exclusive role (DMER) constraints. The importance of SMER constraints and DMER constraints is illustrated by the fact that they are the only types of constraints included in the ANSI/NIST standard for RBAC [ANSI 2004; Ferraiolo et al. 2001].

SSoD policies are *objectives* that need to be achieved. They exist independent of whether RBAC is used to manage the access permissions or not. Each SSoD policy specifies the minimum number of users that are allowed to together possess all permissions for a sensitive task. On the other hand, SMER constraints are *mechanisms* used to achieve SSoD policies. These constraints are specific to RBAC. Each constraint limits the role memberships a single user is allowed to have. Whether a set of SMER constraints enforces a given SSoD policy depends upon how permissions are assigned to roles. For example, if all permissions that are needed to complete a sensitive task are assigned to a single role, SMER constraints cannot ensure that no single user possess all those permissions. Note that the distinction between mechanisms and objectives is a relative one. While SMER constraints are mechanisms to enforce SSoD policies, they may be implemented using other, lower-level mechanisms. At the same time, a SSoD policy, which specifies a lower-bound on the sizes of the sets of users that possess all permissions to complete a sensitive task, may be viewed as a mechanism to ensure that at least a certain number of users is involved for any instance of the sensitive task.

In the literature, this distinction between SSoD policy objectives and SMER constraints as a mechanism to enforce them is sometimes not clearly made. This is evident in the way these constraints are referred to in the literature. SMER constraints are often called Static SoD constraints, and DMER are called Dynamic SoD constraints. For example, in the ANSI RBAC standard [ANSI 2004], SMER constraints are called SSD constraints, and DMER constraints are called DSD constraints. As we discuss in Section 2.2, DMER constraints are not suitable for enforcing SoD policies. In fact, DMER constraints are motivated by the Least Privilege principle rather than the Separation of Duty principle. A

danger with equating SMER constraints with SoD policies is that the SMER constraints may be specified without a clear specification of what objectives they are intended to meet; consequently, it is unclear whether the higher-level objectives are met by the constraints or not. Another danger with equating SMER constraints with SoD policies is that even though when SMER constraints are specified there exist a clear understanding of what SoD policies are desired, when the assignment of permissions to roles changes, the SMER constraints may no longer be adequate for enforcing the desired SSoD policies.

When we make a clear distinction between objectives (SSoD policies) and mechanisms (SMER constraints), several interesting problems arise. For example, the *verification* problem is: "does a set of SMER constraints indeed enforce an SSoD policy?", and the *generation* problem is: "how do we generate a set of constraints that is adequate to enforce an SSoD policy?" Although the use of SMER constraints to support SoD has been studied for over a decade, and several authors (e.g., [Kuhn 1997; Sandhu et al. 1996]) have recognized that SMER constraints are a mechanism to enforce SoD policies, surprisingly these problems have not been examined in the literature, to the best of our knowledge.

1.1 Contributions and organization

Our contributions in this paper are as follows.

—We provide precise definitions for SSoD policies and SMER constraints, and precise formulations for the verification and generation problems. Through these definitions, we clearly illustrate the difference between SSoD policies as policy objectives and SMER constraints as mechanisms.

—We show that directly enforcing SSoD policies in RBAC is intractable ($\mathbf{coNP}$-complete), while enforcing SMER constraints is efficient; however, the verification problem is intractable ($\mathbf{coNP}$-complete), even for a basic subcase of the problem. We also show that the verification reduces naturally to the satisfiability problem (SAT), for which there exist algorithms that have been proven to work well in practice [Du et al. 1997]. We discuss the implications of these results.

—We define 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. For other kinds of SSoD policies, we present an efficient algorithm that generates sets of SMER constraints that minimally enforce the policies.

The results reported here are fundamental to understand the effectiveness of using constraints to enforce high-level SoD policies in RBAC. The verification and generation algorithms are also of practical significance in RBAC systems that use SMER constraints to enforce SSoD policies.

The remainder of the paper is organized as follows. We discuss related work in the next section. In Section 3, we give definitions of SSoD policies and SMER constraints, as well as the computational complexities for enforcing them. In Section 4, we study the verification problem. In Sections 5 and 6, we study the generation problem. We conclude with Section 7.

2. RELATED WORK

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 [Saltzer and Schroeder 1975] under the name "separation of privilege." They credited Roger Needham with making the following observation in 1973: "a protection mechanism that requires two keys to unlock it is more robust and flexible than one that requires only a single key. No single accident, deception, or breach of trust is sufficient to compromise the protected information."

Clark and Wilson's commercial security policy for integrity [Clark and Wilson 1987] identified SoD along with well-formed transactions as two major mechanisms for controlling fraud and error. The use of well-formed transactions ensures that information within the computer system is internally consistent. Separation of duty ensures that the objects in the physical world are consistent with the information about these objects in the computer system. As Clark and Wilson [Clark and Wilson 1987] explained: "Because computers do not normally have direct sensors to monitor the real world, computers cannot verify external consistency directly. Rather, the correspondence is ensured indirectly by separating all operations into several subparts and requiring that each subpart be executed by a different person."

Sandhu [Sandhu 1990; 1988] presented Transaction Control Expressions, a history-based mechanism for dynamically enforcing SoD policies. Nash and Poland [Nash and Poland 1990] 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 data item. In the latter, users are constrained a-priori from performing certain steps. Foley [Foley 1997] proposed a framework based on relabel policies [Foley et al. 1996] to express dynamic SoD requirements. Sandhu and Jajodia [Sandhu and Jajodia 1990] studied what mechanisms are required in a general-purpose multiuser database management system (DBMS) to facilitate the integrity objectives of information systems. They identified SoD as a "timed-honored principle for prevention of fraud and errors, going back to the very beginning of commerce" and explained SoD as "Simply stated, no single individual should be in a position to misappropriate assets on his own. Operationally, this means that a chain of events which affects the balance of assets must require different individuals to be involved at key points, so that without their collusion the overall chain cannot take effect."

2.1 SoD and RBAC

In one of the earliest papers on RBAC, Ferraiolo and Kuhn [Ferraiolo and Kuhn 1992] used the terms Static and Dynamic SoD to refer to static and dynamic enforcement of SoD. In a subsequent paper, Ferraiolo et al. [Ferraiolo et al. 1995] defined Static SoD as: "A user is authorized as a member of a role only if that role is not mutually exclusive with any of the other roles for which the user already possesses membership." This is the requirement of SMER constraints and not an SoD policy. Similarly, Dynamic SoD was defined as forbidding a user from activating roles that are mutually exclusive. We call these DMER constraints. As we argue in Section 2.2, DMER constraints are motivated by the least privilege principle rather than the SoD principle. The NIST standard for RBAC [ANSI 2004; Ferraiolo et al. 2001] adopts the same terminology as Ferraiolo et al. [Ferraiolo et al. 1995].

This distinction between SoD policies as objectives and SMER constraints as a mechanisms has been recognized in the literature. Simon and Zurko [Simon and Zurko 1997] stated "Separation of Duty is a security principle used to formulate multi-person control policies, requiring that two or more different people be responsible for the completion of a

task or a set of related tasks."

Sandhu et al. [Sandhu et al. 1996] stated, in their widely cited paper that introduced the highly influential RBAC96 family of RBAC models, "The most common RBAC constraint is mutually exclusive roles. The same user can be assigned to at most one rule in a mutually exclusive set. This supports separation of duties, which is further ensured by a mutual exclusion constraint on permission assignment." This clearly shows that SMER constraints are not SoD policies themselves but rather mechanisms that can be used to enforce SoD policies. Sandhu et al. [Sandhu et al. 1996] pointed out that constraints on permission assignment can be used together with SMER constraints to better enforce SoD policies.

This distinction is made even clearer by Kuhn [Kuhn 1997], whose work has the title "*Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems*". Kuhn's work [Kuhn 1997] is perhaps the closest in spirit to this paper. Kuhn discussed mutual exclusion of roles for separation of duty and proposed a safety condition: that no one user should possess the privilege to execute every step of a task, thereby being able to complete the task. Our definition for safety in Section 3.1 is a generalization of Kuhn's definition [Kuhn 1997]: setting k to 2 gives us Kuhn's definition. Kuhn [Kuhn 1997] studied the kinds of constraint mechanisms on permission assignment that are needed in addition to SMER constraints to achieve SoD. One such mechanism is complete exclusion in permission assignment, which requires that, if two roles r_1 and r_2 are declared to be mutually exclusive by a SMER constraint, then a privilege that is assigned to one of r_1 and r_2 cannot be assigned to any other role in the RBAC system. Kuhn [Kuhn 1997] showed that SMER constraints together with complete exclusion are sufficient to enforce the special case of SSoD policies he considered. This conclusion is only about the power of the particular constraint mechanism; it says nothing about whether a particular constraint configuration enforces a set of SSoD policies, or how such constraints may be generated. In this paper, in addition to the problem of whether SMER constraints are sufficient to enforce SSoD policies, we consider such verification and constraint generation problems that are not studied by Kuhn [Kuhn 1997].

There also exists a wealth of literature [Ahn and Sandhu 1999; 2000; Crampton 2003; Gligor et al. 1998; Jaeger 1999; Jaeger and Tidswell 2001; Simon and Zurko 1997; Tidswell and Jaeger 2000] on constraints other than SMER and DMER constraints in the context of RBAC. They either proposed and classified new kinds of constraints [Gligor et al. 1998; Simon and Zurko 1997] or proposed new languages for specifying sophisticated constraints [Ahn and Sandhu 1999; 2000; Crampton 2003; Jaeger and Tidswell 2001; Tidswell and Jaeger 2000]. Most of the proposed constraints are variants of SMER and DMER constraints, for example, one may declare two permissions to be mutually exclusive, so that no role or user can be authorized for both permissions, or that two users are mutually exclusive, so that they cannot be assigned to the same role. The study of verification and generation problems related to those more sophisticated constraints is beyond the scope of this paper.

There has also been recent interest in static and dynamic constraints to enforce separation of duty in workflow systems. Atluri and Huang [Atluri and Huang 1996] proposed an access control model for workflow environment, which supports temporal constraints. Bertino et al. [Bertino et al. 1999] proposed a language for specifying static and dynamic constraints for separation of duty in role-based workflow systems. They also discussed an algorithm for the problem of consistency checking in workflows; that is, to check whether a

workflow with such constraints has a valid user-to-task assignment. Botha and Eloff [Botha and Eloff 2001] introduced the "coAP administration paradigm" to express separation of duty constraints in workflow environments, and asserted that it is the first work that allows for the specification of constraints among users, permissions, roles and tasks. They also presented a prototype implementation. Crampton et al. [Crampton 2004; 2005; Tan et al. 2004] considered the consistency checking problem in workflows. Apart from constraints to enforce separation of duty, they also considered other goals, such as binding of duty. A binding of duty constraint mandates that if a particular user is assigned to perform a certain task, then he must also be assigned to perform a certain other task in a workflow instance. Knorr and Stormer [Knorr and Stormer 2001] proposed SSoDL (Simple SoD Language) to specify constraints. A constraint in SSoDL takes the form $(s_1, t_1) \not\rightarrow (s_2, t_2)$, which means that if a subject s_1 has performed task t_1, then s_2 must not perform task t_2. Such constraints can be used to dynamically enforce SoD policies. Kandala and Sandhu [Kandala and Sandhu 2002] proposed to use Transaction Control Expressions [Sandhu 1988] to enforce SoD policies dynamically. We point out that none of the work on constraints in workflow systems has considered the verification and generation problems for constraints that we consider in this paper. The problems considered there are how to enforce the constraints proposed there and whether the constraints are consistent with each other; it is generally assumed in such work that constraints are already specified to meet some implicit objective.

2.2 DMER Constraints

DMER constraints are introduced in [Ferraiolo et al. 1995] under the name DSoD constraints. A DMER constraint prevents a user from simultaneously activating mutually exclusive roles in a session. DMER constraints are called DSoD constraints because they are the "dynamic" version of SMER constraints, which are referred to as SSoD constraints in [Ferraiolo et al. 1995]. However, DMER constraints do not seem to enforce SoD policies at all, because even though they limit the roles a user can activate in a single session, they do not prevent a single user from activating mutually exclusive roles across multiple sessions and finishing a sensitive task on his own. In RBAC, each session has only one user. Thus, a sensitive task that requires multiple users to complete cannot be completed in one session; several sessions are required. For example, suppose that two roles r_1 and r_2 are declared to be dynamically mutually exclusive in a DMER constraint; presumably because in order to complete a sensitive task, one has to combine permissions assigned to r_1 with permissions assigned to r_2. As each session can have only one user, this task cannot be finished in any single session, and multiple sessions are needed to complete the task. A user can thus start a session, activate r_1, use the permissions of r_1 to work on the task, end the session, start another session, activate r_2, and use the permissions of r_2 to finish the task. This does not violate the DMER constraint, but clearly violates the intended SoD policy.

In fact, DMER constraints are motivated by the least privilege principle rather than the SoD principle. The least privilege principle mandates that "every program and every user of the system should operate using the least set of privileges necessary to complete the job" [Saltzer and Schroeder 1975]. By requiring certain roles to be not activated at the same time, one can limit the privileges that a user may use in a session. This aspect is also identified in the ANSI/NIST standard in RBAC: "DSD properties provide extended support for the principle of least privilege in that each user has different levels of permission at different times, depending on the task being performed."

k-n SSoD policy	k-out-of-n Static Separation of Duty policy Notation 1 in Section 3.1
t-m SMER constraint	t-out-of-m Static Mutually Exclusive Roles Notation 5 in Section 3.2
k-n RSSoD requirement	k-out-of-n Role-based Static Separation of Duty Notation 14 in Section 5.2
the SC-SSoD problem	the Safety Checking problem for SSoD policies Definition 4 in Section 3.1
the SC-SMER problem	the Satisfaction Checking problem for SMER constraints Definition 7 in Section 3.2
the EV problem	the Enforcement Verification problem Definition 10 in Section 4
the CEV problem	the Canonical Enforcement Verification problem Definition 12 in Section 4.1

Table I. List of acronyms used in the paper, what they stand for, and where they are defined.

SMER constraints and DMER constraints are the only types of constraints that are included in the ANSI/NIST standard for RBAC [ANSI 2004; Ferraiolo et al. 2001], because they are considered to be the most fundamental and widely supported kinds of constraints. As we discuss above, DMER constraints cannot enforce SoD, we focus on SMER constraints in the rest of this paper.

3. STATIC SEPARATION OF DUTY AND MUTUALLY EXCLUSIVE ROLES

In this section, we give precise definitions for SSoD policies, RBAC, and SMER constraints. Table I lists the acronyms we use in this paper.

Users and *permissions* are at the core of an access control system. The state of an access control system specifies the set of permissions each user has. In this paper, we treat permissions as if they are opaque, i.e., we do not consider the internal structure of permissions. We assume also that permissions are not related to one another, e.g., the possession of one or more permissions does not imply the possession of another permission.

3.1 Static Separation of Duty (SSoD) policies

We now formally define SSoD policies.

NOTATION 1. A k-n SSoD (k-out-of-n Static Separation of Duty) policy is expressed as

$$\mathsf{ssod}\langle\{p_1, \ldots, p_n\}, k\rangle$$

where each p_i is a permission and n and k are integers such that $1 < k \leq n$.

Intuitively, the policy $\mathsf{ssod}\langle\{p_1, \ldots, p_n\}\, k\rangle$ means that at least k users are required to perform a task that requires all these permissions. In other words, there should not exist a set of fewer than k users that together have all the permissions in $\{p_1, \ldots, p_n\}$.

The permissions in a k-n SSoD policy are the permissions needed to carry out a sensitive task, and the policy guarantees that at least k users are needed to successfully complete it. If one wants to specify an SSoD policy, one should first identify a sensitive task, then identify the permissions needed to complete the task, and finally determine the minimum number of collaborating users authorized to complete it.

We now introduce the notion of an RBAC state. We assume that there are three countably infinite sets: $\mathcal{U}$ (the set of all possible users), $\mathcal{R}$ (the set of all possible roles), and $\mathcal{P}$ (the set of all possible permissions).

NOTATION 2. An RBAC state γ is a 3-tuple $\langle UA, PA, RH\rangle$, in which the user assignment relation $UA \subset \mathcal{U} \times \mathcal{R}$ associates users with roles, the permission assignment relation $PA \subset \mathcal{R} \times \mathcal{P}$ associates roles with permissions, and the role hierarchy relation $RH \subset \mathcal{R} \times \mathcal{R}$ specifies a relation among roles.

The reflexive, transitive closure of RH (denoted by RH^*) is a partial order among roles in $\mathcal{R}$. When $(r_1, r_2) \in RH^*$, we often write $r_1 \geq r_2$ and say that r_1 is senior to r_2, which means that every user who is a member of r_1 is also a member of r_2, and that every permission associated with r_2 is also associated with r_1.

We use $\mathsf{Roles}[PA]$ to denote the set of roles that appear in PA, and $\mathsf{Roles}[RH]$ to denote the set of roles that appear in RH.

An RBAC state $\gamma = \langle UA, PA, RH\rangle$ determines the set of roles of which a user is a member, and the set of permissions for which a user is authorized. Formally, γ is associated with two functions, $\mathsf{auth\_roles}_\gamma\colon \mathcal{U} \to 2^{\mathcal{R}}$ and $\mathsf{auth\_perms}_\gamma\colon \mathcal{U} \to 2^{\mathcal{P}}$, where $2^{\mathcal{R}}$ is the powerset of $\mathcal{R}$, and $2^{\mathcal{P}}$ is the powerset of $\mathcal{P}$. The two functions are defined as follows:

$$\begin{aligned}\mathsf{auth\_roles}_\gamma[u] &= \{\, r \in \mathcal{R} \mid \exists r_1 \in \mathcal{R}\,[\,(u, r_1) \in UA \ \wedge\ (r_1, r) \in RH^*\,]\,\}\\ \mathsf{auth\_perms}_\gamma[u] &= \{\, p \in \mathcal{P} \mid \exists r_1, r_2 \in \mathcal{R}\,[\,(u, r_1) \in UA \ \wedge\\ &\qquad\qquad (r_1, r_2) \in RH^* \ \wedge\ (r_2, p) \in PA\,]\,\}\end{aligned}$$

As $\mathsf{auth\_roles}_{\langle UA,PA,RH\rangle}[u]$ is determined only by UA and RH, we sometimes write $\mathsf{auth\_roles}_{(UA,RH)}[u]$.

DEFINITION 3. We say that an RBAC state γ is *safe* with respect to an SSoD policy $e = \mathsf{ssod}\langle\{p_1, \ldots, p_n\}, k\rangle$, which we denote by $safe_e(\gamma)$, if and only if in state γ no $k-1$ users together have all the permissions in the policy. More precisely,

$$safe_e(\gamma) \ \triangleq\ \forall u_1 \cdots u_{k-1} \in \mathcal{U}\ \left(\left(\bigcup_{i=1}^{k-1} \mathsf{auth\_perms}_\gamma[u_i]\right) \not\supseteq \{p_1, \ldots, p_n\}\right).$$

An RBAC state γ is *safe* with respect to a set E of SSoD policies, which we denote by $safe_E(\gamma)$, if and only if γ is safe with respect to every policy in the set.

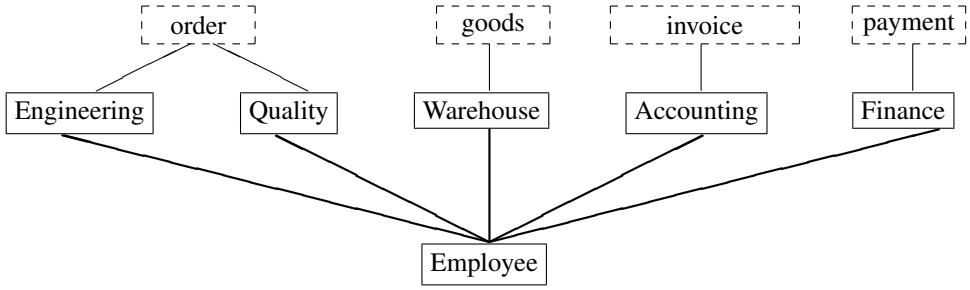
DEFINITION 4. SC-SSOD (the Safety Checking problem for SSoD policies) is defined to be the following problem: Given an RBAC state γ and a set E of SSoD policies, determine if $safe_E(\gamma)$ is true.

Observe that if no $k-1$ users together have all the permissions in a policy, then no set of fewer than k users together have all the permissions.

EXAMPLE 1. Consider the task of ordering and paying for goods discussed in Section 1. We have a permission corresponding to each step in the task; these permissions are p_{order}, $p_{invoice}$, p_{goods}, and $p_{payment}$. We have the following set of SSoD policies:

$$\begin{aligned}E_1 &= \{e_1, e_2\}\\ e_1 &= \mathsf{ssod}\langle\{p_{order}, p_{invoice}, p_{goods}, p_{payment}\}, 3\rangle\\ e_2 &= \mathsf{ssod}\langle\{p_{order}, p_{payment}\}, 2\rangle\end{aligned}$$

Consider the RBAC state $\gamma_1 = \langle UA_1, PA_1, RH_1\rangle$, where $UA_1 = \{$(Alice, Warehouse), (Alice, Finance), (Bob, Accounting), (Bob, Quality),



$RH_1 = \{$ (Engineering, Employee), (Quality, Employee), (Warehouse, Employee), (Accounting, Employee), (Finance, Employee) $\}$.
$PA_1 = \{$ (Engineering, p_{order}), (Quality, p_{order}), (Warehouse, p_{goods}), (Accounting, $p_{invoice}$), (Finance, $p_{payment}$) $\}$.

Fig. 1. A sample role hierarchy and permission assignment. Roles are shown in solid boxes, and permissions in dashed boxes. A line segment represents either a role-role relationship, or the assignment of a permission to a role.

(Carl, Engineering)$\}$, and PA_1 and RH_1 are given in Figure 1. The state γ_1 is not safe with respect to e_1, a 3-4 SSoD policy, as the 2 users Alice and Bob together possess all 4 permissions in e_1.

Given a set E of SSoD policies, suppose an RBAC system starts at a state that is safe with respect to E. Each time one is about to make a change to the system that may affect safety, one checks whether the RBAC state that results from the proposed change is safe and makes the change only when the answer is affirmative. Such a change may be the addition of a new user-role assignment to UA, a new role-permission assignment to PA, or a new pair to RH. This approach to ensuring that an RBAC system is safe requires solving SC-SSoD, which turns out to be intractable.

THEOREM 1. SC-SSoD *is* **coNP**-*complete.*

PROOF. Consider the complement of SC-SSoD, i.e., given an RBAC state γ and a set E of SSoD policies, determine if $safe_E(\gamma)$ is false, which is denoted by $\overline{\text{SC-SSoD}}$. It suffices to show that $\overline{\text{SC-SSoD}}$ is **NP**-complete.

We first show that $\overline{\text{SC-SSoD}}$ is in **NP**. If an RBAC state γ is not safe wrt. E, then there exists a k-n SSoD policy in E and $k-1$ users such that in γ these $k-1$ users together have the n permissions in the SSoD policy. If one correctly guesses the k-n SSoD policy and the $k-1$ users that together have all the n permissions in the policy, verifying that the guess is correct can be done in polynomial time: compute the union of the $k-1$ users' permissions and check whether it is a superset of the set of permissions in the SSoD policy.

We now show that $\overline{\text{SC-SSoD}}$ is **NP**-hard by reducing the set covering problem to it. In the set covering problem, the inputs are a finite set $\mathcal{S}$, a family $F = \{S_1, \ldots, S_\ell\}$ of subsets of $\mathcal{S}$, and a budget B. The goal is to determine whether there exist B sets in F whose union is $\mathcal{S}$. This problem is **NP**-complete [Garey and Johnson 1979; Papadimitriou

1994].

The reduction is as follows. Given $\mathcal{S}$, F, and B, construct an SSoD policy e as follows: Let each element in $\mathcal{S}$ map to a permission in the policy, let k be $B+1$ and let n be the size of $\mathcal{S}$. We have constructed a k-n SSoD policy $\mathsf{ssod}\langle S, B+1\rangle$. Construct an RBAC state γ as follows. For each corresponding permission in $\mathcal{S}$, create a role to which the permission is assigned. For each different subset S_i ($1 \leq i \leq \ell$) in F, create a user u_i to which all roles in S_i are assigned. The resulting RBAC state γ is not safe with respect to $\{e\}$ if and only if B sets in F cover $\mathcal{S}$. $\square$

In the reduction in the proof, each permission is assigned to one role and the role hierarchy relation is empty; thus the problem remains **coNP**-complete even when we restrict ourselves to the case of RBAC without a role hierarchy.

While SC-SSoD is in in general intractable, efficient algorithms for SC-SSoD exist when all the SSoD policies in E have small k. For example, when checking whether γ is safe with respect to a 2-n SSoD policy, one only needs to compute the set of permissions of every user and check whether it is a superset of the permissions in the policy. This has a worst-case time complexity of $O(N_u(N_r + N_p))$, where N_u is the number of users in γ, N_r the number of roles, and N_p the number of permissions. More generally, a straightforward algorithm for solving SC-SSoD for a k-n SSoD policy has running time grows polynomially in the number of users and roles and exponentially only in k.

3.2 Statically mutually exclusive role (SMER) constraints

In RBAC, constraints such as mutually exclusive roles are introduced to enforce SSoD policies. In the most basic form, two roles may be declared to be mutually exclusive in the sense that no user is allowed to be a member of both roles. We now present a generalized form of such constraints.

NOTATION 5. A t-m SMER (t-out-of-m Statically Mutually Exclusive Role) constraint is expressed as

$$\mathsf{smer}\langle\{r_1, \ldots, r_m\}, t\rangle$$

where each r_i is a role, and m and t are integers such that $1 < t \leq m$. Such a constraint is said to be *canonical* of *cardinality* t when $t = m$.

DEFINITION 6. We say that an RBAC state γ *satisfies* a t-m SMER constraint $c = \mathsf{smer}\langle\{r_1, \ldots, r_m\}, t\rangle$, which we denote by $satisfies_c(\gamma)$, if and only if no user is a member of t or more roles in $\{r_1, \ldots, r_m\}$. More formally,

$$satisfies_c(\gamma) \triangleq \forall u \in \mathcal{U} \quad (|\, \mathsf{auth\_roles}_\gamma[u] \cap \{r_1, \ldots, r_m\} \,| < t)\,.$$

When γ does not satisfy a SMER constraint, we say that γ *violates* the SMER constraint. An RBAC state γ *satisfies* a set C of SMER constraints, which we denote by $satisfies_C(\gamma)$, if and only if γ satisfies every constraint in the set.

DEFINITION 7. SC-SMER (*the Satisfaction Checking problem for SMER constraints*) is defined to be the following problem: Given an RBAC state γ and a set C of SMER constraints, determine whether γ satisfies C.

We point out that our notion of $\mathsf{auth\_roles}_\gamma$ (for an RBAC state, γ) from Notation 2 incorporates the role hierarchy. That is, the user u is a member of the role r in state

γ if and only if $r \in \mathsf{auth\_roles}_\gamma[u]$. This definition for SMER constraints is consistent with the definition for static separation of duty constraints from the ANSI standard for RBAC [ANSI 2004].

Each SMER constraint restricts the role memberships of a single user, in contrast to a k-n SSoD policy, which restricts the permissions possessed by a set of $k-1$ users. Therefore, there is an efficient algorithm to check whether an RBAC state γ satisfies a set C of SMER constraints.

THEOREM 2. SC-SMER *is in* $\mathbf{P}$.

PROOF. One algorithm for solving SC-SMER is as follows. For each t-m SMER constraint in C and for each user in γ, one first computes the set of all roles the user is a member of, then counts how many roles in this set also appear in the set of roles in the SMER constraint, and finally compares this number with t. This algorithm has a time complexity of $O(N_u N_r M)$, where N_u is the number of users in γ, N_r the number of roles in γ, and M is the number of constraints. □

Given an RBAC state γ that satisfies a set C of SMER constraints, in order to ensure that the state that results from adding a new user-role assignment to γ still satisfies C, one needs to only check the role memberships of that user, which can be done in time $O(N_r M)$. This is more efficient than solving the SC-SSoD problem each time an assignment is added to ensure that the state that results is safe with respect to the SSoD policies.

4. THE ENFORCEMENT VERIFICATION PROBLEM

The facts that SC-SSoD is intractable and that an efficient algorithm exists for SC-SMER appear to provide a justification for using SMER constraints to enforce SSoD policies. However, when using SMER constraints to enforce SSoD policies, one needs to answer the question whether a set of SMER constraints is adequate to enforce a set of SSoD policies. The answer to this question depends on what permissions each role has, which is determined by the permission assignment PA and the role hierarchy RH. For instance, if all permissions in an SSoD policy are assigned to one role, then no set of SMER constraints enforces that policy.

NOTATION 8. An *SSoD configuration* is a 3-tuple $\langle E, PA, RH\rangle$, where E is a set of SSoD policies, PA is a permission assignment relation, and RH is a role hierarchy.

DEFINITION 9. We say that a set C of SMER constraints *enforces* the SSoD configuration $\langle E, PA, RH\rangle$ if and only if the following is true.

$$\forall UA \subset \mathcal{U} \times \mathcal{R}\ [satisfies_C(\langle UA, PA, RH\rangle) \Rightarrow safe_E(\langle UA, PA, RH\rangle)]$$

DEFINITION 10. EV (*the Enforcement Verification problem*) is defined to be the following problem: Given an SSoD configuration $\langle E, PA, RH\rangle$ and a set C of SMER constraints, determine whether C enforces $\langle E, PA, RH\rangle$.

If C enforces an SSoD configuration $\langle E, PA, RH\rangle$, then no matter how the user assignment relation UA changes, as long as it results in an RBAC state that satisfies C, the state is safe with respect to the SSoD policies in E.

EXAMPLE 2. Continuing from Example 1, we consider the following set of SMER constraints

$$\begin{aligned} C_1 &= \{c_1, c_2, c_3\} \\ c_1 &= \mathsf{smer}\langle\{\mathsf{Warehouse}, \mathsf{Accounting}, \mathsf{Finance}\},\ 2\rangle \\ c_2 &= \mathsf{smer}\langle\{\mathsf{Engineering}, \mathsf{Finance}\},\ 2\rangle \\ c_3 &= \mathsf{smer}\langle\{\mathsf{Quality}, \mathsf{Finance}\},\ 2\rangle \end{aligned}$$

The constraint c_1 ensures that no user is a member of any two roles in Warehouse, Accounting, and Finance; thus, the smallest number of users that have memberships in all the three roles is three, and therefore, the smallest number of users that the permissions p_{goods}, $p_{invoice}$, and $p_{payment}$ is also three. This ensures safety with respect to the SSoD policy e_1. The constraints c_2 and c_3 together ensure safety with respect to e_2. Thus C_1 enforces $\langle E_1, PA_1, RH_1\rangle$.

In Example 1, we observed that the state γ_1 is not safe with respect to E_1; therefore, it does not satisfy C_1. In particular, γ_1 violates the constraint c_1 because Alice is assigned to both Warehouse and Accounting.

We now establish an upper bound on the computational complexity of EV.

LEMMA 3. EV *is in* **coNP**.

PROOF. Consider the complement problem of EV, denoted by $\overline{\text{EV}}$, i.e., determine whether C does not enforce $\langle E, PA, RH\rangle$. This problem is essentially one of determining whether there exists a user-role assignment such that $satisfies_C(\langle UA, PA, RH\rangle)$ is true, but $safe_E(\langle UA, PA, RH\rangle)$ is false. It suffices to show that $\overline{\text{EV}}$ is in **NP**. To do this, we show that if C does not enforce $\langle E, PA, RH\rangle$, then a short (polynomial in the input size) evidence exists that can be verified in polynomial time.

If a set C of t-m SMER constraints does not enforce $\langle E, PA, RH\rangle$, then there exists a counter-example, i.e., a user-role assignment UA such that $satisfies_C(\langle UA, PA, RH\rangle)$ is true but $safe_E(\langle UA, PA, RH\rangle)$ is false. That is, there exists a k-n SSoD policy in E that is violated by $k-1$ users. If such an UA exists, then a subset of the UA that consists of just the $k-1$ users is also a counter-example. Thus, the smallest counter-example has size linear in the size of the input. Once the counter-example is guessed, its correctness can be verified in time polynomial in the size of the input. This shows that $\overline{\text{EV}}$ is in **NP**. □

4.1 A special case of the EV problem

In this section, we show that every set of SMER constraints can be equivalently represented using a set of canonical (t-t) SMER constraints. Therefore, we need to consider only such constraints. We then study the enforcement verification problem for canonical SMER constraints.

DEFINITION 11. Let C_1 and C_2 be two sets of SMER constraints. We say that C_1 and C_2 are *equivalent* if and only if for every RBAC state γ, γ satisfies C_1 if and only if γ satisfies C_2.

Clearly, if C_1 and C_2 are equivalent, then C_1 enforces $\langle E, PA, RH\rangle$ if and only if C_2 enforces $\langle E, PA, RH\rangle$.

LEMMA 4. *For every t-m SMER constraint c, there exists a set C' of canonical SMER constraints of cardinality t such that C' and $\{c\}$ are equivalent.*

PROOF. Given a t-m SMER constraint $c = \langle \{r_1, \ldots, r_m\}, t \rangle$, where $m > t$. Let C' be

$$\{\ \mathsf{smer}\langle R, t\rangle \ \mid\ R \subset \{r_1, \ldots, r_m\} \wedge |R| = t\ \}.$$

That is, C' is the set of all constraints $\mathsf{smer}\langle R, t\rangle$ such that R is a size-t subset of $\{r_1, \ldots, r_m\}$. It is easy to see that the violation of any constraint in C' implies the violation of the constraint c and the violation of the constraint c implies the violation of some constraint in C'. Therefore, C' and $\{c\}$ are equivalent. □

It follows from Lemma 4 that for every set C of SMER constraints, there exists a set C' of canonical SMER constraints such that C and C' are equivalent with respect to the allowed configurations. Furthermore, given an instance of EV in which the set E contains more than one SSoD policy, one can verify these policies one by one. Therefore, without loss of generality, we assume that E is a singleton set, i.e., $E = \{e\}$ consists of one policy. This enables us to limit our attention to the following special case of EV.

DEFINITION 12. CEV (the Canonical Enforcement Verification problem) is defined as follows: Given PA, RH, a singleton set $\{e\}$ of SSoD policies and a set C of canonical SMER constraints, determine whether C enforces $\langle \{e\}, PA, RH\rangle$.

An algorithm solving CEV can be used to solve EV, as any EV instance can be translated into a set of CEV instances. However, the resulting CEV instance may have an exponential blowup in size, as we would have $\binom{m}{t}$ canonical SMER constraints for each t-m SMER constraint. On the other hand, if an RBAC system uses only canonical constraints to start with, then such blowup does not occur.

4.2 Computational complexities of CEV and EV

We now show that CEV is $\mathbf{coNP}$-hard by showing that a special case of it is $\mathbf{coNP}$-complete. The special case we consider is whether a set of 2-2 SMER constraints satisfies a 2-n SSoD policy. Recall that a 2-2 SMER constraint specifies that two roles are mutually exclusive, i.e., no user can be a member of both roles. This is the most common kind of constraints considered in the literature. A 2-n SSoD specifies that no single user is allowed to possess all of n given permissions. This is the simplest and most common kind of SSoD policy. This special case is thus arguably the simplest verification problem.

THEOREM 5. *Determining whether a set of* 2-2 *SMER constraints enforces* $\langle \{e\}, PA, RH\rangle$*, where* e *is a* 2-n *SSoD policy, is* $\mathbf{coNP}$*-complete.*

PROOF. That this problem is in $\mathbf{coNP}$ follows from Lemma 3.

We prove that this problem is $\mathbf{coNP}$-hard by reducing MONOTONE-3-2-SAT to the complement of this problem. We define MONOTONE-3-2-SAT as being a special case of monotone-CNF-SAT where each clause is composed of either three positive literals or two negative literals. We show in Appendix A that MONOTONE-3-2-SAT is $\mathbf{NP}$-complete.

Given a MONOTONE-3-2-SAT problem instance composed of positive clauses of the form $(v_{i_1} \vee v_{i_2} \vee v_{i_3})$, $1 \leq i \leq n$, and negative clauses $(\neg v_{j_1} \vee \neg v_{j_2})$, $1 \leq j \leq m$, we do the following reduction: (1) Each propositional variable v is mapped to a role $r(v)$. Intuitively, that v is assigned true means that a user u is assigned to be a member of $r(v)$. (2) Each positive clause $(v_{i_1} \vee v_{i_2} \vee v_{i_3})$ is mapped to a permission p_i assigned to the three roles $r(v_{i_1})$, $r(v_{i_2})$, and $r(v_{i_3})$. Intuitively, the clause is true if and only if the user u has permission p_i. (3) Each negative clause $(\neg v_{j_1} \vee \neg v_{j_2})$ is mapped to a 2-2 SMER

constraint $\mathsf{smer}\langle\{r(v_{j_1}), r(v_{j_2})\}, 2\rangle$. This negative clause is true if and only if the user is not a member of both $r(v_{j_1})$ and $r(v_{j_2})$.

The MONOTONE-3-2-SAT instance is satisfiable if and only if the 2-n SSoD policy $\mathsf{ssod}\langle\{p_1, p_2, ..., p_n\}, 2)\rangle$ can be violated while satisfying all the 2-2 SMER constraints. □

COROLLARY 6. EV *and* CEV *are* **coNP**-*complete.*

PROOF. Follows directly from Lemma 3 and Theorem 5. □

4.3 Solving CEV using SAT

It is easier to think about the complement of CEV, denoted by $\overline{\text{CEV}}$: If C does not enforce an SSoD configuration $\langle\{\mathsf{ssod}\langle\{p_1, \cdots, p_n\}, k\rangle\}, PA, RH\rangle$, then there exists a user-to-role assignment UA such that all the SMER constraints in C are satisfied by the state $\langle UA, PA, RH\rangle$ but there are $k-1$ users together possess all permissions $\{p_1, \cdots, p_n\}$. In the rest of this section, we show that the $\overline{\text{CEV}}$ naturally reduces to SAT. An implication of the existence of such a reduction is that we can use algorithms for SAT to solve CEV. Given a CEV instance, the answer is yes if and only if the corresponding SAT instance is not satisfiable. SAT has been studied extensively for several decades (see, for example, [Du et al. 1997]). Many clever algorithms exist that can answer most instances efficiently. The fact that $\overline{\text{CEV}}$ naturally reduces to SAT means that one can benefit from the extensive research on SAT to provide practical enforcement checking.

An instance of the $\overline{\text{CEV}}$ problem is given by PA, RH, a set C of canonical constraints, and a k-n SSoD policy e. We need to map such an instance to a SAT instance such that the SAT instance is satisfiable if and only if C does not enforce $\langle\{e\}, PA, RH\rangle$. In other words, if the SAT instance is satisfiable, then we can find a user assignment relation UA such that the constraints in C are satisfied but the state $\langle UA, PA, RH\rangle$ is not safe with respect to e.

We first give such a mapping for a subcase of $\overline{\text{CEV}}$ where e is a 2-n SSoD policy, i.e., no single user has all n permissions in e. When constructing a SAT instance from such a $\overline{\text{CEV}}$ instance, our goal is to find a user-to-role assignment for one single user such that this user has all permissions in e without violating any constraint in C.

The SAT instance is constructed as follows. For each role r appearing in PA, RH, C, create a propositional variable v_r. Intuitively, if v_r is true, then the user is a member of the role v_r. Construct the set of clauses for the SAT instance as follows.

—For each permission p in e, if $r'_1, r'_2, \cdots, r'_\ell$ are all the roles that are associated with the permission p, then add the clause

$$v_{r'_1} \vee v_{r'_2} \vee \cdots \vee v_{r'_\ell}$$

This clause means that, to have the permission p, the user must be a member of one of the roles that are associated with the permission p.

—For each constraint $c = \mathsf{smer}\langle\{r_1, r_2, \cdots, r_t\}, t\rangle$ in C, add to the instance the following clause

$$\neg v_{r_1} \vee \neg v_{r_2} \vee \cdots \vee \neg v_{r_t}$$

This clause means that, to satisfy the constraint, there must be at least one role in $\{r_1, r_2, \cdots, r_t\}$ of which the user is not a member.

—For each role hierarchy relationship $(r_1, r_2) \in RH$, add to the instance the following clause

$$\neg v_{r_1} \vee v_{r_2}$$

This clause means that if a user is a member of r_1, then it must also be a member of r_2.

If the SAT instance is satisfiable, let I be a truth assignment that makes the instance true, then assign the user to be a member of every role r such that v_r is true in I. The user has all permissions in e without violating any constraint in C; therefore, C does not enforce $\langle\{e\}, PA, RH\rangle$. On the other hand, if C does not enforce $\langle\{e\}, PA, RH\rangle$, then there exists a UA such that in the RBAC state $\langle UA, PA, RH\rangle$ there exists a user who has all permissions in e, and the role memberships of the user in this state give rise to a truth assignment that make the SAT instance true.

We now give the mapping for any instance of $\overline{\text{CEV}}$. Given a k-n SSoD policy e where $k > 2$, we need to consider role memberships of $k-1$ different users at the same time. Our goal is to find a user assignment relation such that $k-1$ together have all permissions in e, yet none of the $k-1$ user's role memberships violate constraints in C.

Given PA and RH, let C be a set of canonical constraints and e be a k-n SSoD policy. Construct a SAT instance as follows. For each role the appears in PA, RH and C, create $k-1$ proposition variables. The propositional variables created for a role r is denoted $v_r^1, v_r^2, \cdots, v_r^{k-1}$. Intuitively, v_r^i is true when the i^{th} user is a member of the role r. Then construct the set of clauses for the SAT instance as follows.

—For each permission p in e, if $r'_1, r'_2, \cdots, r'_\ell$ are all the roles that are associated with the permission p, then add the clause

$$v_{r'_1}^1 \vee \cdots \vee v_{r'_1}^{k-1} \ \vee \ v_{r'_2}^1 \vee \cdots \vee v_{r'_2}^{k-1} \ \vee \cdots \vee \ v_{r'_\ell}^1 \vee \cdots \vee v_{r'_\ell}^{k-1}$$

To have the permission p, at least one of the $k-1$ users must be a member of one of these roles.

—For each constraint $c \in C$, let $c = \mathsf{smer}\langle\{r_1, r_2, \cdots, r_t\}, t\rangle$; for each i from 1 to $k-1$, add the following clause

$$\neg v_{r_1}^i \vee \neg v_{r_2}^i \vee \cdots \vee \neg v_{r_t}^i$$

To satisfy the constraint, for every user, there must exist a role in $\{r_1, r_2, \cdots, r_t\}$ of which the user is not a member.

—For each role hierarchy relationship $(r_1, r_2) \in RH$, and for each i from 1 to $k-1$, add the following clause

$$\neg v_{r_1}^i \vee v_{r_2}^i$$

This clause means that if a user is a member of r_1, then it must also be a member of r_2.

The SAT instance is satisfiable if and only if C does not enforce $\langle\{e\}, PA, RH\rangle$.

4.4 Comparing approaches for enforcing SSoD policies

In Section 3.1 we discussed one approach for enforcing SSoD policies. Each time one is about to make a change to the system that may affect safety, one checks whether the RBAC state that results from the proposed change is safe and makes the change only when the answer is affirmative. We call this approach direct enforcement; it requires solving SC-SSoD.

One can also use SMER constraints for enforcing SSoD policies; we call this indirect enforcement. This requires solving SC-SMER. This approach also requires one to ensure that existing SMER constraints enforce the desired SSoD policies. One way to achieve this to by solving EV. Another way to achieve this is to generating SMER constraints from SSoD policies with the guarantee that the generated constraints always enforce the policies. We will show how to perform constraint generation in Sections 5 and 6.

In summary, we have three approaches for enforcing SSoD policies:

(1) direct enforcement by solving SC-SSoD,

(2) indirect enforcement by solving SC-SMER and EV, and

(3) indirect enforcement by solving SC-SMER and the constraint generation problem.

We now compare these approaches.

Recall that SC-SSoD is intractable ($\mathbf{coNP}$-complete) in general; however it is efficiently solvable when the value k in a k-n SSoD policy is small. SC-SMER is efficiently solvable; however, EV is intractable ($\mathbf{coNP}$-complete). In fact, we have shown that verifying whether a set of 2-2 SMER constraints enforces $\langle\{e\}, PA, RH\rangle$, where e is a 2-n SSoD policy, may be intractable. This result is quite surprising, and casts doubts on the effectiveness of the approach of using SMER constraints to enforce SSoD policies, which has been adopted in the literature without being questioned for years. From computation complexity point of view, we can conclude the following. For SSoD policies with smaller k value, approach (1) is more advantageous than approach (2). For more general SSoD policies, approach (3) is the most advantageous if one can solve the constraint generation problem efficiently.

Going beyond complexity class in comparing approaches (1) and (2), we make two observations that favor approach (2).

First, when using SMER constraints to enforce SSoD policies, EV, which can be computationally expensive, needs to be performed only when either a new role-role relationship is added to the role hierarchy or a permission in an SSoD policy is assigned to some role. When a user is assigned to a role, only constraint checking (SC-SMER) needs to be performed, which is quite efficient. On the other hand, when enforcing SSoD policies directly, the expensive safety checking (SC-SSoD) needs to be performed every time a user is assigned to a role of which the user was not already a member. As user-to-role assignment is the most dynamic relation, enforcing SSoD policies directly may be overall more expensive than using SMER constraints.

Second, the complexity of SC-SSoD is calculated in the number of users plus the number of roles, and the complexity of CEV is calculated in the number of roles only. (In both cases, one needs to consider only the permissions in the SSoD policies, rather than all permissions in the RBAC state.) As most RBAC systems have many more users than roles, solving EV is likely to be more efficient than solving SC-SSoD in practice. For example, in a case study of the RBAC system of an European bank [Schaad et al. 2001], it has been reported that there are about 40000 users and 1300 roles, with a role/user ratio of approximately 3.2%. The study [Schaad et al. 2001] also cites an oral estimate that was given at the RBAC2000 workshop, suggesting that the number of roles in a role-based system is approximately 3-4% of the user population.

Given the above comparisons, we can see that both approach (1) and approach (2) have advantages over each other, and approach (3) seems to be the most promising approach.

More conclusive comparisons among them require access to data in real-world systems and are beyond the scope of this paper. To decide which approach to use in a given situation, one would need to know the kinds of SSoD policies and constraints that one is likely to encounter. Results in this paper provide helpful information in making the decision.

5. CONSTRAINT GENERATION: FROM SSOD POLICIES TO RSSOD REQUIREMENTS

Section 4 considers the problem of verifying that SMER constraints in RBAC enforce the desired SSoD policies. In this section and the next section we study the problem of generating a set of SMER constraints that are adequate for enforcing SSoD policies. The generation problem that we study is as follows: Given an SSoD configuration $\langle E, PA, RH\rangle$, where E is a set of SSoD policies, PA is a permission assignment relation, and RH is a role hierarchy, generate a set C of SMER constraints that enforces $\langle E, PA, RH\rangle$. Such a set C ensures that for every possible user assignment relation UA, as long as all constraints in C are satisfied, the RBAC state is guaranteed to be safe with respect to E. This way, when the UA relation is changed, one only needs to check whether constraints in C remain satisfied.

In Section 5.1, we study the problem of determining whether an SSoD configuration is enforceable. In Section 5.2, we introduce the notion of Role-level Static Separation of Duty (RSSoD) requirements as an intermediate step from SSoD policies to SMER constraints. In Section 5.3, we study the computational complexity of several problems related to RSSoD requirements.

5.1 Enforceability of SSoD configurations

Given an SSoD configuration $\langle E, PA, RH\rangle$, there may not exist a set C of constraints that enforces $\langle E, PA, RH\rangle$.

DEFINITION 13. An SSoD configuration $\langle E, PA, RH\rangle$ is *enforceable* if and only if there exists a set C of SMER constraints such that C enforces $\langle E, PA, RH\rangle$.

LEMMA 7. *An SSoD configuration $\langle E, PA, RH\rangle$ is **not** enforceable if and only if there exist an SSoD policy $\mathsf{ssod}\langle\{p_1, \cdots, p_n\}, k\rangle$ in E and $k-1$ roles in PA such that these $k-1$ roles together have all the permissions in $\{p_1, \cdots, p_n\}$ and none of these $k-1$ roles is senior to a role other than itself.*

PROOF. For the "only if" part, we show that if the condition in the lemma does not exist then the SSoD configuration is enforceable. Consider the set C of constraints that includes a 2-2 SMER constraint for every pair of roles in PA and RH; this means that every pair of roles is mutually exclusive. This forbids any user from being assigned to any role that is senior to a role other than itself. For example, if $r_1 \geq r_2$, then to satisfy the constraint $\mathsf{smer}\langle\{r_1, r_2\}, 2\rangle$, no user can be a member of r_1. In any state that satisfies these constraints, only those roles that are not senior to any role other than itself can have any member. Furthermore, no user is a member of two roles. Therefore, $k-1$ users together can be member of at most $k-1$ roles. If the condition in the lemma does not exist, then no $k-1$ users have all permissions in $\{p_1, \cdots, p_n\}$. Therefore, C enforces the SSoD configuration.

For the "if" part, consider the following. If there exist an SSoD policy $\mathsf{ssod}\langle\{p_1, \cdots, p_n\}, k\rangle$ in E and $k-1$ roles in PA such that these $k-1$ roles together

have all the permissions in $\{p_1, \cdots, p_n\}$ and none of these $k-1$ roles is senior to a role other than itself, then one can construct an RBAC state in which there are $k-1$ users and each of the user is assigned one of the $k-1$ role. Such a state is unsafe with respect to E. In such an RBAC state, each user is a member of only one role, thus no SMER constraint is violated. Therefore, no matter which set of SMER constraints one specifies, the state will satisfy the constraints, while being unsafe. □

We observe from the proof of the above lemma that given any enforceable configuration, 2-2 SMER constraints are sufficient to enforce it.

THEOREM 8. *Determining whether an SSoD configuration is enforceable is* **coNP***-complete.*

PROOF. We consider the complement of the problem, that is, to determine whether an SSoD configuration is *not* enforceable. We first show that this problem is in **NP**: From Lemma 7 given a solution set of $k-1$ roles we compute the set of permissions assigned to those roles, and check whether it is a superset of the set of permissions in the SSoD policy.

We show that the problem is **NP**-hard by reducing the set covering problem, which is **NP**-complete, to it. In the set covering problem, the inputs are a finite set $\mathcal{S}$, a family $F = \{S_1, \ldots, S_\ell\}$ of subsets of $\mathcal{S}$, and a budget B. The goal is to determine whether there exists B sets in F whose union is $\mathcal{S}$. Given an instance of the set covering problem: $\mathcal{S}$, F, and B, construct an SSoD policy e as follows. Let each element in $\mathcal{S}$ map to a permission in the policy $\mathsf{ssod}\langle S, B+1\rangle$, let k be $B+1$, and let n be the size of $\mathcal{S}$. We have constructed a k-n SSoD policy. For each different subset S_i ($1 \leq i \leq \ell$) in F, create a role r_i and assign to it all permissions corresponding to the elements in S_i. B sets in F cover $\mathcal{S}$ if and only if the SSoD configuration is not enforceable. □

Similar to SC-SSOD, efficient algorithms exist when all the SSoD policies in the configuration have small k.

5.2 RSSoD requirements

SSoD policies are expressed in terms of restrictions on permissions. On the other hand, SMER constraints are expressed in term of restrictions on role memberships. In order to generate SMER constraints for enforcing SSoD policies, the first step is to translate restrictions on permissions expressed in SSoD policies to restrictions on role memberships. We now define such role-level SSoD requirements.

NOTATION 14. A k-n RSSoD (k-out-of-n Role-based Static Separation of Duty) requirement has the form

$$\mathsf{rssod}\langle\{r_1, \ldots, r_n\}, k\rangle$$

where each r_i is a role and n and k are integers such that $1 < k \leq n$.

When a set of users together have memberships in every role in a set of roles, we say that these users *cover* the roles in the set.

DEFINITION 15. We say that an RBAC state γ is *safe* with respect to the RSSoD requirement $d = \mathsf{rssod}\langle\{r_1, \ldots, r_n\}, k\rangle$, denoted by $safe_d(\gamma)$, if and only if there does not exist a set of fewer than k users that cover the n roles in d. More formally,

$$safe_d(\gamma) \triangleq \forall u_1 \cdots u_{k-1} \in \mathcal{U} \left(\left(\bigcup_{i=1}^{k-1} \mathsf{auth\_roles}_\gamma[u_i] \right) \not\supseteq \{r_1, \ldots, r_n\} \right).$$

An RBAC state γ is *safe* with respect to a set D of RSSoD requirements, denoted by $safe_D(\gamma)$, if and only if it is safe with respect to every requirement in D.

We now show that, given an SSoD configuration $\langle E, PA, RH\rangle$, one can generate a set D of RSSoD requirements from E and PA such that D captures the security restrictions inherent in the configuration.

DEFINITION 16. Given PA and E, *the standard set of RSSoD requirements derived from PA and E*, denoted by $D^S(E, PA)$, is defined as follows:

—if there exist $\mathsf{ssod}\langle P, k\rangle \in E$ and $R \in 2^{\mathcal{R}}$ such that $(|R| < k) \wedge (\mathsf{auth\_perms}_{PA}[R] \supseteq P)$, then $D^S(E, PA) = \emptyset$, where $\emptyset$ is the empty set, and $\mathsf{auth\_perms}_{PA}[R] = \{p \mid (r, p) \in PA \wedge r \in R\}$ gives the set of all permissions that are assigned to some role in R according to PA;

—otherwise, $D^S(E, PA)$ includes $\mathsf{rssod}\langle R, k\rangle$ for each $\mathsf{ssod}\langle P, k\rangle \in E$ and each $R \in 2^{\mathcal{R}}$ such that $(\mathsf{auth\_perms}_{PA}[R] \supseteq P) \wedge \forall R' \subset R\, (\mathsf{auth\_perms}_{PA}[R'] \not\supseteq P)$.

In other words, $D^S(E, PA)$ can be computed as follows: for each $\mathsf{ssod}\langle P, k\rangle \in E$, and for each minimal set of roles that have all permissions in P according to PA, declare that no $k-1$ users can cover all roles in the set. If any of the set of roles so computed has size smaller than k, then set $D^S(E, PA)$ to $\emptyset$. We point out $D^S(E, PA) = \emptyset$ if and only if either $\langle E, PA, RH\rangle$ is not enforceable (see Lemma 7), or $\langle UA, PA, RH\rangle$ is trivially safe (that is, some permission from P is not assigned to any role).

The following theorem shows that for any SSoD configuration $\langle E, PA, RH\rangle$, the set $D^S(E, PA)$ captures the security requirement in the configuration.

THEOREM 9. *Given E and PA, the set $D^S(E, PA)$ satisfies the following condition.*

$$\forall UA\ \forall RH\ \left[safe_E(\langle UA, PA, RH\rangle) \Leftrightarrow safe_{D^S(E,PA)}(\langle UA, PA, RH\rangle)\right]$$

PROOF. Given any UA and RH, if $\gamma = \langle UA, PA, RH\rangle$ is not safe with respect to $D^S(E, PA)$, then there is an RSSoD requirement $\mathsf{rssod}\langle R, k\rangle$ in $D^S(E, PA)$ and there exist $k-1$ users in γ such that these users together are authorized for all roles in R. Given the way in which $D^S(E, PA)$ is defined, there exists an SSoD policy in E such that the roles in R together have all permissions in it; therefore γ is not safe with respect to E.

If $\gamma = \langle UA, PA, RH\rangle$ is not safe with respect to E, then there exist $e = \mathsf{ssod}\langle\{p_1, \ldots, p_n\}, k\rangle$ and $k-1$ users that together have all permissions in $\{p_1, \ldots, p_n\}$. Let R_1 be the set of all roles that these users are members of in γ, and let R_2 be the set of roles in PA, then the set of permissions that are authorized to $R_1 \cap R_2$ includes $\{p_1, \ldots, p_n\}$. By the definition of $D^S(E, PA)$, there exists an RSSoD requirement $d = \mathsf{rssod}\langle R, k\rangle$ such that $R \subseteq R_1 \cap R_2$; γ is not safe with respect to d; and thus not safe with respect to $D^S(E, PA)$. □

EXAMPLE 3. Consider the SSoD configuration given in Example 1, the following set of RSSoD requirements precisely captures the security requirements in the configuration.

$$\begin{aligned}
D_1 &= \{d_1, d_2, d_3, d_4\}\\
d_1 &= \mathsf{rssod}\langle\{\mathsf{Engineering}, \mathsf{Warehouse}, \mathsf{Accounting}, \mathsf{Finance}\}, 3\rangle\\
d_2 &= \mathsf{rssod}\langle\{\mathsf{Quality}, \mathsf{Warehouse}, \mathsf{Accounting}, \mathsf{Finance}\}, 3\rangle\\
d_3 &= \mathsf{rssod}\langle\{\mathsf{Engineering}, \mathsf{Finance}\}, 2\rangle\\
d_4 &= \mathsf{rssod}\langle\{\mathsf{Quality}, \mathsf{Finance}\}, 2\rangle
\end{aligned}$$

5.3 Runtime Checking and Verification for RSSoD Requirements

Our main objective to introduce RSSoD requirements is to use them as an intermediate step from SSoD policies to SMER constraints. Several problems naturally come up, and we now study their computational complexities.

DEFINITION 17. The Runtime RSSoD Checking Problem is as follows: Given an RBAC state γ and a set D of RSSoD requirements, determine whether $safe_D(\gamma)$.

THEOREM 10. *The Runtime RSSoD Checking Problem is* $\mathbf{coNP}$*-complete.*

PROOF. The proof is essentially the same as that for Theorem 1, by reducing the set-covering problem to the complement of the Runtime RSSoD Checking Problem. The universe $\mathcal{S}$ is mapped to the set of all roles in the RSSoD requirement and each subset S_i ($1 \leq i \leq \ell$) in F is mapped to the set of roles assigned to a user. □

DEFINITION 18. The RSSoD Verification Problem is as follows: Given PA, RH, a set D of RSSoD requirements, and a set C of SMER constraints, determine whether C enforces D (under PA and RH), that is, whether

$$\forall UA \left[satisfies_{\langle UA,PA,RH \rangle}(C) \Rightarrow safe_{\langle UA,PA,RH \rangle}(D) \right]$$

The RSSoD Verification Problem can be viewed as a special case of the Enforcement Verification Problem in which the role hierarchy is empty and the permission assignment is degenerated (i.e., each permission is assigned to a different role). Because the permission assignment is degenerated, the hardness of this problem does not follow from the coNP-completeness of the problem to determine whether a set of 2-2 SMER constraints enforces a 2-n SSoD policy; the proof there involves a reduction from MONOTONE-3-2-SAT that uses permission assignment in a non-degenerated way.

THEOREM 11. *The problem of verifying whether a set of* 2-2 *SMER constraints enforces a* 4-n *RSSoD requirement is* $\mathbf{coNP}$*-complete.*

PROOF. We consider the complement of the problem: determine whether there exists a user-role assignment such that the SMER constraints are satisfied and $k-1$ users together are authorized for n roles in the RSSoD requirement. We need to show that the complement is $\mathbf{NP}$-complete.

That the complement problem is in $\mathbf{NP}$ follows from the fact that the EV problem is in $\mathbf{coNP}$. (Lemma 3)

To show $\mathbf{NP}$-hardness, we use a reduction from the GRAPH 3-COLORABILITY problem [Garey and Johnson 1979], which determines whether a graph G is 3-colorable. Given a graph with n vertices, we map each vertex in the graph to a role, and each edge in the graph to a 2-2 SMER constraint where the two roles are the ones corresponding to the two endpoints of the edge. Let $r_1, \cdots, r_n$ be all roles that result from this process and let C be the set of resultant constraints. A 4-n RSSoD requirement d is constructed as $\mathsf{rssod}\langle \{r_1, \ldots, r_n\}, 4 \rangle$. We now show that C does not enforce $\{d\}$ if and only if the graph is 3-colorable. Each user is viewed as a color, and a node is assigned a color means that the user is a member of the role corresponding to the node. The graph is 3-colorable if and only if there are 3 users who can cover all n roles without violating any of the constraints. □

6. CONSTRAINT GENERATION: FROM RSSOD REQUIREMENTS TO SMER CONSTRAINTS

In this section, we show how to generate SMER constraints from a set of RSSoD requirements. In Section 5.2, we have shown how to compute a set of RSSoD requirements that capture the security requirements inherent in an SSoD configuration $\langle E, PA, RH\rangle$. Together these sections show how one can generate SMER constraints to enforce an SSoD configuration.

As there are often multiple sets of constraints that can enforce the same set of SSoD policies, we would like to generate constraint sets that are not "more restrictive than necessary". For this purpose, we formally define the notion of "precise enforcement" and "minimal enforcement". In Section 6.1, we show that in two cases one can generate SMER constraints that precisely enforce RSSoD requirements and that precise enforcement is impossible in other cases. In Section 6.2, we show that while 2-2 SMER constraints are sufficient to enforce any RSSoD requirements, having k-k SMER constraints for larger k enables more precise enforcement. In Section 6.3, we show how to generate all singleton constraint sets that minimally enforce an RSSoD requirement.

6.1 Precise enforcement of RSSoD requirements

One way to enforce a set of RSSoD requirements is to declare every pair of roles to be mutually exclusive. However, this is often times too restrictive. Ideally, we want to generate constraints that precisely enforce the RSSoD requirements.

DEFINITION 19. Let D be a set of RSSoD requirements and C be a set of SMER constraints, we say that C *enforces* D if and only if the following holds:

$$\forall \text{ RBAC state } \gamma \ [satisfies_C(\gamma) \Rightarrow safe_D(\gamma)]$$

We say that C *is necessary to enforce* D if and only if the following holds:

$$\forall \text{ RBAC state } \gamma \ [safe_D(\gamma) \wedge live_D(\gamma) \Rightarrow satisfies_C(\gamma)]$$

where $live_D(\gamma)$ means that for every role r appearing in D, there exists a user who is a member of r.
We say C *precisely enforces* D if and only if C both enforces D and is necessary to enforce D.

EXAMPLE 4. To understand the reason that $live_D(UA, RH)$ is needed in Definition 19, consider the following example.

$$\begin{aligned} D &= \{\mathsf{rssod}\langle\{r_1, r_2, r_3\}, 3\rangle\} \\ C &= \{\mathsf{smer}\langle\{r_1, r_2, r_3\}, 2\rangle\} \end{aligned}$$

The RSSoD requirement means that at least 3 users are required to cover the 3 roles $\{r_1, r_2, r_3\}$. To enforce this, one would naturally require that no single user is a member of 2 or more roles in $\{r_1, r_2, r_3\}$, which is the constraint in C. Intuitively, C is necessary to enforce E. However, $\forall\gamma \ [safe_D(\gamma) \Rightarrow satisfies_C(\gamma)]$ is not true, as there exist RBAC states that violate C and are safe with respect to D. Consider $RH = \emptyset$ and $UA = \{(u_1, r_1), (u_1, r_2)\}$. The constraint in C is violated because u_1 is a member of both r_1 and r_2. Yet (UA,RH) is safe with respect to D, as no user is a member of r_3, and thus no 2 users cover $\{r_1, r_2, r_3\}$. However, if we extend the RBAC state to make some user a

member of r_3, then the resulting state is not safe with respect to D. The extra condition $live_D(UA, RH)$ addresses this issue.

We now give two cases where precise enforcement is possible.

LEMMA 12. *Given a k-k RSSoD requirement* $d = \mathsf{rssod}\langle\{r_1, \cdots, r_k\}, k\rangle$*, the singleton constraint set* $\{c = \mathsf{smer}\langle\{r_1, \cdots, r_k\}, 2\rangle\}$ *precisely enforces* $\{d\}$.

PROOF. The requirement d means that k users are required to cover all k roles. The constraint c means that no user is allowed to be a member of 2 roles in the set. We first show that $\{c\}$ enforces $\{d\}$. If no user is a member of 2 roles from the set of k roles, then k users are needed to cover the k roles. To show that c is necessary, consider the following. Given an RBAC state γ that violates c, we show that $live_{\{d\}}(\gamma)$ and $safe_{\{d\}}(\gamma)$ cannot both be true. As γ violates c, there exists a user who has memberships in 2 roles from the set of k roles. If $live_{\{d\}}(\gamma)$ is true, then for every role other than the 2 roles there exists a user who is a member of it. Thus $k-1$ users cover the k roles, and $safe_{\{d\}}(\gamma)$ is false. □

LEMMA 13. *Given a 2-n RSSoD requirement* $d = \mathsf{rssod}\langle\{r_1, \cdots, r_n\}, 2\rangle$*, the singleton constraint set* $\{\, c = \mathsf{smer}\langle\{r_1, \cdots, r_n\}, n\rangle \,\}$ *precisely enforces* $\{\, d \,\}$.

PROOF. The requirement d means that 2 users are required to cover all n roles. The constraint c means that no user is allowed to be a member of all the roles in the set. We first show that $\{\, c \,\}$ enforces $\{\, d \,\}$. If no user is allowed to be authorized for all n roles from the set, then at least 2 users are required to cover all the n roles. To show that $\{\, c \,\}$ is necessary, consider the following. Given an RBAC state γ that violates c, there is a user that is a member of all roles from the set of n roles. Thus $safe_{\{d\}}(\gamma)$ is false. □

In fact, as we prove in Lemma 21 (in Section 6.3 after results needed for the proof have been developed), for every k and n such that $2 < k < n$, there exists no set of SMER constraints that precisely enforces a k-n RSSoD requirement. That is, the two special cases in Lemmas 12 and 13 are the only cases where precise enforcement can be achieved. As precise enforcement is not achievable in many cases, we give a method to generate "good" sets of SMER constraints that are as precise as possible.

6.2 Expressive power of different t-m SMER constraints

Before discussing the generation of "good" sets of SMER constraints, we look at the expressive power of t-m SMER constraints using different values of t and m. We would like to answer questions such as: Does an RBAC system that supports 3-3 SMER constraints have more expressive power than an RBAC system that supports only 2-2 SMER constraints? Answers to such questions will help developers of RBAC systems decide which kinds of constraints to support.

From Lemma 4 we know that t-m SMER constraints, where $m > t$, can be equivalently represented using t-t SMER constraints; thus non-canonical constraints do not add additional expressive power in terms of enforcing SSoD policies. From the proof of Lemma 7, we know that 2-2 SMER constraints are sufficient for enforcing (albeit not always precisely) any enforceable SSoD configuration. We now show that 2-2 SMER constraints (or 2-n SMER constraints which can be equivalently expressed using 2-2 SMER constraints) are required in the sense that they cannot be replaced with other k-n SMER (where $k \geq 3$) constraints.

LEMMA 14. *There exist RSSoD requirements that cannot be enforced without using 2-n SMER constraints.*

PROOF. A t-t RSSoD requirement $\mathsf{rssod}\langle\{r_1, \cdots, r_t\}, t\rangle$ requires that no $t-1$ users cover all t roles. It can be enforced only by using 2-n SMER constraints, as these are the only type of constraints that prevent two roles from being assigned to a single user. □

Although in all cases, 2-2 SMER constraints are sufficient to enforce enforceable SSoD configurations, using only such constraints is not always desirable as other kinds of constraints may provide more precise enforcement. The following lemma expresses this.

LEMMA 15. *For any $n > 2$, there exists an RSSoD requirement that can be precisely enforced using a canonical constraint of cardinality n but cannot be precisely enforced using any set of t-m SMER constraints with $t < n$.*

PROOF. Consider the 2-n RSSoD requirement $d = \mathsf{rssod}\langle\{r_1, \cdots, r_n\}, 2\rangle$ (at least 2 users are required to cover the n roles). The n-n SMER constraint $c = \mathsf{smer}\langle\{r_1, \cdots, r_n\}, n\rangle$ (no single user is allowed to be authorized for all n roles) precisely enforces the configuration, as was shown in lemma 13.

We now show that no set of t-m SMER constraints with $t < n$ precisely enforces $\{\, d \,\}$. Assume, for the sake of contradiction, that there exists such a set. Then there exists a set C of canonical constraints of cardinalities less than n that also precisely enforces $\{\, d \,\}$. At least one constraint, c, in C must be such that all roles in the constraint are in $\{r_1, \cdots, r_n\}$; otherwise, one could assign one user to have all roles in $\{r_1, \cdots, r_n\}$ without violating any constraint in C. Because c is a canonical constraint of cardinality $t < n$, the set S of roles in c is a strict subset of $\{r_1, \cdots, r_n\}$. This constraint c is not necessary for implementing the SSoD configuration, as an RBAC state in which a user is assigned to be a member of all roles in S is safe with respect to the requirement d, as long as the user is not a member of some role in $\{r_1, \cdots, r_n\} - S$. □

This lemma suggests that if one wants to enforce an arbitrary RSSoD requirement as precisely as possible, then one needs to support n-n SMER constraints for arbitrary n.

6.3 "Good" sets of SMER constraints: Definition

While two sets of constraints may both enforce the same RSSoD requirements, they have different degrees of restrictiveness.

DEFINITION 20. Let C_1 and C_2 be two sets of SMER constraints. We say that C_1 is *at least as restrictive as* C_2 (denoted by $C_1 \succeq C_2$) if and only if the following holds:

$$\forall \text{ RBAC state } \gamma \ \left[\ \mathit{satisfies}_{C_1}(\gamma) \Rightarrow \mathit{satisfies}_{C_2}(\gamma)\ \right].$$

The $\succeq$ relation among all sets of SMER constraints is a partial order. When $C_1 \succeq C_2$ but not $C_2 \succeq C_1$, we say that C_1 is *more restrictive than* C_2 (denoted by $C_1 \succ C_2$). By definition, C_1 and C_2 are *equivalent* (Definition 11) if and only if $C_1 \succeq C_2$ and $C_2 \succeq C_1$.

When neither $C_1 \succeq C_2$ nor $C_2 \succeq C_1$, we say C_1 and C_2 are *incomparable*.

When both C and C' enforce a set D of RSSoD requirements, there are four cases:

(1) $C \succ C'$, in which case C' is preferable to C for enforcing D as it is less restrictive (and thus more precise);

(2) $C' \succ C$, in which C is preferable to C';

(3) C and C' are equivalent, in which case either C or C' can be used;
(4) C and C' are incomparable, in which case the decision to choose C over C' (or C' over C) depends on considerations other than SSoD policies.

DEFINITION 21. Given a set D of RSSoD requirements, we say that a set C of SMER constraints is *minimal* for enforcing D if and only if C enforces D and there does not exist a different set C' of SMER constraints such that C' also enforces D and $C \succ C'$ (C is more restrictive than C').

Our approach to dealing with the generation problem is to exhaustively generate all singleton sets of SMER constraints that are minimal for enforcing D (for any such set, no other set is strictly more preferable to it based on precision of enforcement) and leave the choice to the system administrator.

LEMMA 16. *If a set C of SMER constraints precisely enforces a set D of RSSoD requirements, then for any C' that also enforces D, $C' \succeq C$, i.e., C' is at least as restrictive as C.*

PROOF. Given C that precisely enforces D and C' that also enforces D, we need to show that: (a) $\forall\gamma(satisfies_{C'}(\gamma) \Rightarrow satisfies_{C}(\gamma))$.

From the fact that C' enforces D, we have (by Definition 19) $\forall\gamma(satisfies_{C'}(\gamma) \Rightarrow safe_{D}(\gamma))$. From the fact that C precisely enforces D, we know that C is necessary for enforcing D. By Definition 19, we have $\forall\gamma(live_{D}(\gamma) \wedge safe_{D}(\gamma) \Rightarrow satisfies_{C}(\gamma))$. Combining the two, we have: (b) $\forall\gamma(live_{D}(\gamma) \wedge satisfies_{C'}(\gamma)) \Rightarrow satisfies_{C}(\gamma))$.

Note that (a) and (b) differ in that (b) has $live_{D}(\gamma)$ in the antecedent inside the range of the quantifier $\forall\gamma$. We now show that (b) implies (a). Suppose, for the sake of contradiction, that (b) is true but (a) is not. Then there exists a state γ such that $satisfies_{C'}(\gamma)$ is true, but $satisfies_{C}(\gamma)$ is not. If such a state γ exists, then there exists a state γ_1 such that the role hierarchy relation in γ_1 is empty, and $satisfies_{C'}(\gamma_1)$ is true, but $satisfies_{C}(\gamma_1)$ is false. The state γ_1 can be constructed by computing all role memberships in γ and assigning these role memberships using UA. We then construct a state γ_2 by adding to γ_1 the following user-to-role assignments: for each role r mentioned in D assign a new user (one who is not a member of any role in γ_1) to be a member of r. (Different users are used for different roles, so each new user is a member of exactly one role.) We denote the resultant state γ_2. Clearly, $live_{D}(\gamma_2)$ is true. Furthermore, by construction of γ_2, $satisfies_{C'}(\gamma) \Leftrightarrow satisfies_{C'}(\gamma_2)$, and $satisfies_{C}(\gamma) \Leftrightarrow satisfies_{C}(\gamma_2)$. Therefore, $satisfies_{C'}(\gamma_2)$ is true but $satisfies_{C}(\gamma_2)$ is not. This is in contradiction to (b) being true. □

LEMMA 17. *Let C be a set of SMER constraints that precisely enforces a set D of RSSoD requirements. C is minimal for enforcing D. Furthermore, if C_1 is also minimal for enforcing D, then C and C_1 are equivalent.*

PROOF. Given any C' that also enforces D, it follows from Lemma 16 that $C' \succeq C$, thus it cannot be $C \succ C'$ (which implies that $\neg(C' \succeq C)$). This shows that C is minimal.

Given any C_1 that is also minimal for enforcing D, it follows from Lemma 16 that $C_1 \succeq C$. Then either $C_1 \succ C$, or C_1 and C is equivalent. As we have just shown that $C_1 \succ C$ is impossible, C_1 and C must be equivalent. □

LEMMA 18. *Given a set D of RSSoD requirements, if both C_1 and C_2 are minimal for enforcing D and C_1 and C_2 are incomparable, then there exists no set C of SMER constraints that precisely enforces D.*

PROOF. By Contradiction. If there exists a set C that precisely enforces D, then from Lemma 17, C is equivalent to C_1 and to C_2. This contradicts the fact that C_1 and C_2 are incomparable. □

6.4 Generating Minimal Sets of SMER Constraints

Below we define the function SMER-Gen, where SMER-Gen(rssod$\langle R, k\rangle$) gives all singleton sets of SMER constraints that are minimal for enforcing one RSSoD requirement.

$$\text{SMER-Gen}\left(\mathsf{rssod}\langle R, k{=}2\rangle\right) \;=\; \{\,\{\,\mathsf{smer}\langle R, n\rangle\}\,\}$$

$$\text{SMER-Gen}\left(\mathsf{rssod}\langle R, k{\geq}3\rangle\right) \;=\; \bigcup_{j=2}^{\lfloor \frac{n-1}{k-1} \rfloor + 1} \left\{ \{\,\mathsf{smer}\langle R', j\rangle\} \;\middle|\; \begin{array}{c} R' \subseteq R \wedge \\ |R'| = (k{-}1)(j{-}1){+}1 \end{array} \right\}$$

To understand the definition of SMER-Gen, observe the following.

—When given rssod$\langle R, 2\rangle$ (requiring 2 users to cover all roles in R) as the input, SMER-Gen contains one set $\{\mathsf{smer}\langle R, |R|\rangle\}$ (no user can be a member of all roles in R).

—When given rssod$\langle R, |R|\rangle$ (requiring $|R|$ users to cover all roles in R) as the input, we have $t = n = |R|$, and SMER-Gen contains one set $\{\mathsf{smer}\langle R, 2\rangle\}$ (no user can be a member of two roles in R).

—When given rssod$\langle\{r_1, r_2, r_3, r_4\}, 3\rangle$ (requiring 3 users to cover all 4 roles) as the input, SMER-Gen contains four singleton constraint sets. One of them is $\{\mathsf{smer}\langle\{r_1, r_2, r_3\}, 2\rangle\}$, which means that no user can be a member of 2 or more roles in $\{r_1, r_2, r_3\}$, which means that 3 users are required to cover all 3 roles. The other three constraint sets that are generated use other size-3 subsets of $\{r_1, r_2, r_3, r_4\}$. Each of the 4 generated constraint sets is minimal for enforcing the 3-4 RSSoD requirement.

—Given rssod$\langle\{r_1, r_2, r_3, r_4, r_5\}, 3\rangle$ (requiring 3 users to cover all 5 roles) as the input, SMER-Gen contains a total of 11 singleton constraint sets. Ten of them (one for each size-3 subset of the set of roles0 each contains one 2-3 SMER constraint. The 11'th one contains one 3-5 SMER constraint $\mathsf{smer}\langle\{r_1, r_2, r_3, r_4, r_5\}, 3\rangle$, which mandates that each user is a member of at most 2 roles in the set of 5 roles, and is sufficient to ensure that 3 users are required to cover all 5 roles.

EXAMPLE 5. SMER-Gen(rssod$\langle\{\mathsf{Engineering}, \mathsf{Warehouse}, \mathsf{Accounting}, \mathsf{Finance}\}, 3\rangle$) contains the following four singleton sets of SMER constraints:

$$\begin{aligned} C_4 &= \{\,\mathsf{smer}\langle\{\mathsf{Warehouse}, \mathsf{Accounting}, \mathsf{Finance}\},\ 2\rangle\,\} \\ C_5 &= \{\,\mathsf{smer}\langle\{\mathsf{Engineering}, \mathsf{Accounting}, \mathsf{Finance}\},\ 2\rangle\,\} \\ C_6 &= \{\,\mathsf{smer}\langle\{\mathsf{Engineering}, \mathsf{Warehouse}, \mathsf{Finance}\},\ 2\rangle\,\} \\ C_7 &= \{\,\mathsf{smer}\langle\{\mathsf{Engineering}, \mathsf{Warehouse}, \mathsf{Accounting}\},\ 2\rangle\,\} \end{aligned}$$

Any SMER constraint from above is sufficient to satisfy the RSSoD requirement. Each constraint is minimal and the constraints are incomparable with each other. Each leaves a different role unconstrained. For example, if one picks C_5 as the constraint to use, then the role Warehouse is not constrained.

The correctness of this algorithm is justified by the following two lemmas.

LEMMA 19. *Given an RSSoD requirement d, each SMER constraint generated by* `SMER-Gen(d)` *is minimal for enforcing* $\{\,d\,\}$.

PROOF. Let $d = \mathsf{rssod}\langle R_d, k\rangle$ be an k-n RSSoD requirement, and $c = \mathsf{smer}\langle R_c, j\rangle$ be a j-m SMER constraint generated by the algorithm with d as input. Then $n = |R_d|$ and $m = |R_c|$.

By Lemma 4 we know that c can be equivalently expressed as a set C of j-j SMER constraints. Assume, for the sake of contradiction, that $\{ c \}$ is not minimal. Then there exists a set C' of SMER constraints that also enforces $\{ d \}$ and C' is less restrictive than C, i.e., $(C \succeq C') \wedge \neg(C' \succeq C)$.

Because $\neg(C' \succeq C)$, there exists a state γ such that $satisfies_{C'}(\gamma)$ is true but $satisfies_{C}(\gamma)$ is not. This means that at least one j-j SMER constraint $c_v \in C$ is violated by γ. Let R_v be the set of j roles in c_v; there exists a user in γ who is a member of the j roles in R_v. As γ satisfies C', having one user being a member of all roles in R_v does not violate C'.

We now construct another state γ' such that $satisfies_{C'}(\gamma')$ is true, but $safe_{\{d\}}(\gamma')$ is not. This would contradict the assertion that C' enforces $\{ d \}$.

In order to construct γ', we first construct another state γ_1 as follows:

—Use $k-2$ users to cover the roles in $R_c - R_v$ with each user being a member of at most $j-1$ roles in $R_c - R_v$. This is possible for the following reasons. First, $|R_c - R_v| = m - j$. Second, from the algorithm, $m = (k-1)(j-1)+1$. Thus $k-2$ users can cover $(k-2)(j-1)$ roles, where $(k-2)(j-1) = m-j$.

—Assign one of the $k-2$ users to be a member of all roles in $R_d - R_c$.

Observe that γ_1 satisfies $c = \mathsf{smer}\langle R_c, j\rangle$. This is because c does not place any restriction on role memberships about roles in $R_d - R_c$ and each user in γ_1 has at most $j-1$ roles in R_c. Because C is equivalent to $\{c\}$, $satisfies_{C}(\gamma_1)$ is true. Because $C \succeq C'$, $satisfies_{C'}(\gamma_1)$ is also true.

We now construct γ' by adding to γ_1 a new user and assigning the user to be a member of all roles in R_v. The state γ' has $k-1$ users; together they have memberships in all roles in R_d. Thus $safe_{\{d\}}(\gamma')$ is false. The state γ' also satisfies C', as the role memberships of the $k-2$ users in γ_1 do not violate C' and neither does the new user who is a member of all roles in R_v. □

LEMMA 20. *Given an RSSoD requirement d, every SMER constraint that is minimal for enforcing $\{ d \}$ is generated by* `SMER-Gen(d)`.

PROOF. Given an k-n RSSoD requirement $d = \mathsf{rssod}\langle R_d, k\rangle$, we show that any j'-m' SMER $c = \mathsf{smer}\langle R_c, j'\rangle$ that is not generated by the algorithm is not minimal in enforcing $\{ d \}$. We show this using case-by-case analysis for c. In each case, we show that either $\{ c \}$ does not enforce $\{ d \}$, $\{ c \}$ is not minimal in enforcing $\{ d \}$, or $\{ c \}$ is generated by the algorithm. The conditions in the five cases cover all possible situations.

Case 1: $j' > \left\lfloor \frac{n-1}{k-1} \right\rfloor + 1$.

Then $j' \geq \left\lfloor \frac{n-1}{k-1} \right\rfloor + 2$. By assigning to each of $k-1$ users $j'-1$ roles, we are able to cover $(k-1)(j'-1)$ roles without violating the constraint c. Observe that for every pair of positive integers x, y, $\left\lfloor \frac{y}{x} \right\rfloor \geq \frac{y-(x-1)}{x}$. Thus, $(k-1)(j'-1) \geq (k-1)\left(\left\lfloor \frac{n-1}{k-1} \right\rfloor + 1\right) \geq (k-1)\left(\frac{(n-1)-(k-2)}{k-1} + 1\right) = n$. Therefore, $\{ c \}$ does not enforce $\{ d \}$.

Case 2: $j' \leq \left\lfloor \frac{n-1}{k-1} \right\rfloor + 1$ and $R_c \not\subseteq R_d$.

Consider the SMER constraint $c' = \mathsf{smer}\langle j', R_c \cap R_d \rangle$. Clearly, $\{ c \} \succ \{ c' \}$, as violating c' means violating c. Furthermore, $\{ c' \}$ enforces $\{ d \}$ if and only if $\{ c \}$ enforces $\{ d \}$, as any restrictions c has on memberships in roles that appear d also exist in c'. Therefore $\{ c \}$ is not minimal.

Case 3: $j' \leq \left\lfloor \frac{n-1}{k-1} \right\rfloor + 1$, $R_c \subseteq R_d$, and $m' = m$, where $m = (k-1)(j'-1)+1$.

Such a constraint c is generated by the algorithm.

Case 4: $j' \leq \left\lfloor \frac{n-1}{k-1} \right\rfloor + 1$, $R_c \subseteq R_d$, and $m' > m$.

As the algorithm generates a j'-m SMER constraint for each size-m subset of R_d, there exist a constraint c' generated by the algorithm with a set of roles R' such that $R' \subset R_c$. This implies $\{ c' \} \succ \{ c \}$, therefore $\{ c' \}$ is not minimal.

Case 5: $j' \leq \left\lfloor \frac{n-1}{k-1} \right\rfloor + 1$, $R_c \subseteq R_d$, and $m' < m$.

We have $m' \leq (k-1)(j'-1)$. By assigning to $k-1$ users at most $j'-1$ roles each, we are able to cover all m' roles in c without violating c. By further assigning one user to be a member of all roles in $R_d - R_c$, the state is not safe with respect to d while satisfying c. Thus $\{ c \}$ does not enforce $\{ d \}$. □

Lemmas 18 and 19 enable us to prove the following Lemma.

LEMMA 21. *Given a k-n RSSoD requirement where $2 < k < n$, there exists no set of SMER constraints that precisely enforces it.*

PROOF. Assume that $RH = \emptyset$. It suffices to prove that when $2 < k < n$, the algorithm generates at least two SMER constraints, as then, we know that each such constraint is minimal (Lemma 19) and therefore there exists no set of SMER constraints that precisely enforces the RSSoD requirement (Lemma 18).

To show that the algorithm generates at least two SMER constraints when $2 < k < n$, we observe that either (1) $\left\lfloor \frac{n-1}{k-1} \right\rfloor > 1$, or (2) $\left\lfloor \frac{n-1}{k-1} \right\rfloor = 1$. If (1) is true, then j takes on at least the two values 2 and 3 in the outer loop, and therefore the inner loop (lines 7–8) is executed at least twice for different values of j, thereby generating two different SMER constraints.

If (2) is true, then the outer loop (line 5) is executed only once. For that execution of the outer loop, $m = k < n$. Thus there exist more than one size-m subsets of R. Therefore, the inner loop (lines 7–8) executes at least twice for two different size-m subsets R' of R, thereby giving two different SMER constraints. □

Our algorithm does not generate all sets of SMER constraints that are minimal to enforce an RSSoD requirement. Constraint sets of cardinality greater than 1 may exist that are minimal for enforcing the requirement. Our algorithm generates all possible minimal singleton sets of k-m SMER constraints.

7. CONCLUSIONS AND FUTURE WORK

We have posed and answered several fundamental questions related to the use of SMER constraints to enforce SSoD policies, while making a clear distinction between objectives and mechanisms. We have shown that directly enforcing SSoD policies is intractable ($\mathbf{coNP}$-complete), while enforcing SMER constraints is efficient. We have also shown that verifying whether a set of SMER constraints enforces a set of SSoD policies is intractable ($\mathbf{coNP}$-complete), even for a basic subcase of the problem, but reduces naturally

to the satisfiability problem (SAT), for which there exist algorithms that have been proven to work well in practice [Du et al. 1997]. We have discussed why these intractability results should not lead us to conclude that SMER constraints are not an appropriate mechanism for enforcing SSoD policies.

We have defined minimal and precise enforcement. We have also characterized the kinds of policies for which precise enforcement is achievable and shown what constraints precisely enforce such policies. We have also presented an algorithm that generates all singleton SMER constraint sets each of which minimally enforces an RSSoD requirement.

An interesting problem that remains is whether the generation algorithm can be improved to consider preexisting SMER constraints and to consider a set of SSoD policies as a whole rather than individually. Other kinds of constraints also have been proposed for RBAC, e.g., cardinality constraints and constraints on permission assignment [Sandhu et al. 1996]. It would be interesting to examine the use of SMER constraints together with these constraints to enforce SSoD policies.

REFERENCES

AHN, G.-J. AND SANDHU, R. S. 1999. The RSL99 language for role-based separation of duty constraints. In *Proceedings of the 4th Workshop on Role-Based Access Control*. 43–54.

AHN, G.-J. AND SANDHU, R. S. 2000. Role-based authorization constraints specification. *ACM Transactions on Information and System Security 3,* 4 (Nov.), 207–226.

ANSI. 2004. American national standard for information technology – role based access control. ANSI INCITS 359-2004.

ATLURI, V. AND HUANG, W. 1996. An authorization model for workflows. In *Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS)*. 44–64.

BALDWIN, R. W. 1990. Naming and grouping privileges to simplify security management in large databases. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*. 116–132.

BERTINO, E., FERRARI, E., AND ATLURI, V. 1999. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security 2,* 1 (Feb.), 65–104.

BOTHA, R. AND ELOFF, J. 2001. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal 40,* 3, 666–682.

CLARK, D. D. AND WILSON, D. R. 1987. A comparision of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 184–194.

CRAMPTON, J. 2003. Specifying and enforcing constraints in role-based access control. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*. Como, Italy, 43–50.

CRAMPTON, J. 2004. An algebraic approach to the analysis of constrained workflow systems. In *Proceedings of the 3rd Workshop on Foundations of Computer Security*. Turku, Finland, 61–74.

CRAMPTON, J. 2005. A reference monitor for workflow systems with constrained task execution. In *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies (SACMAT 2005)*. Stockholm, Sweden, 38–47.

DU, D., GU, J., AND PARDALOS, P. M., Eds. 1997. *Satisfiability Problem: Theory and Applications*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 35. AMS Press.

FERRAIOLO, D. F., CUIGINI, J. A., AND KUHN, D. R. 1995. Role-based access control (RBAC): Features and motivations. In *Proceedings of the 11th Annual Computer Security Applications Conference (ACSAC'95)*.

FERRAIOLO, D. F. AND KUHN, D. R. 1992. Role-based access control. In *Proceedings of the 15th National Information Systems Security Conference*.

FERRAIOLO, D. F., KUHN, D. R., AND CHANDRAMOULI, R. 2003. *Role-Based Access Control*. Artech House.

FERRAIOLO, D. F., SANDHU, R. S., GAVRILA, S., KUHN, D. R., AND CHANDRAMOULI, R. 2001. Proposed NIST standard for role-based access control. *ACM Transactions on Information and Systems Security 4,* 3 (Aug.), 224–274.

FOLEY, S., GONG, L., AND QIAN, X. 1996. A security model of dynamic labeling providing a tiered approach to verification. In *Proceedings of IEEE Symposium on Research in Security and Privacy*. 142–153.

FOLEY, S. N. 1997. The specification and implementation of 'commercial' security requirements including dynamic segregation of duties. In *Proceedings of the 4th ACM Conference on Computer and Communications Security (CCS-4)*. 125–134.

GAREY, M. R. AND JOHNSON, D. J. 1979. *Computers And Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company.

GLIGOR, V. D., GAVRILA, S. I., AND FERRAIOLO, D. F. 1998. On the formal definition of separation-of-duty policies and their composition. In *Proceedings of IEEE Symposium on Research in Security and Privacy*. 172–183.

JAEGER, T. 1999. On the increasing importance of constraints. In *Proceedings of ACM Workshop on Role-Based Access Control*. 33–42.

JAEGER, T. AND TIDSWELL, J. E. 2001. Practical safety in flexible access control models. *ACM Transactions on Information and System Security 4,* 2 (May), 158–190.

KANDALA, S. AND SANDHU, R. 2002. *Secure Role-Based Workflow Models*. 45–58.

KNORR, K. AND STORMER, H. 2001. *Modeling and Analyzing Separation of Duties in Workflow Environments*. 199–212.

KUHN, D. R. 1997. Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems. In *Proceedings of the Second ACM Workshop on Role-Based Access Control (RBAC'97)*. 23–30.

LI, N., BIZRI, Z., AND TRIPUNITARA, M. V. 2004. On mutually-exclusive roles and separation of duty. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS-11)*. ACM Press, 42–51.

NASH, M. J. AND POLAND, K. R. 1990. Some conundrums concerning separation of duty. In *Proceedings of IEEE Symposium on Research in Security and Privacy*. 201–209.

PAPADIMITRIOU, C. H. 1994. *Computational Complexity*. Addison Wesley Longman.

SALTZER, J. H. AND SCHROEDER, M. D. 1975. The protection of information in computer systems. *Proceedings of the IEEE 63,* 9 (September), 1278–1308.

SANDHU, R. 1990. Separation of duties in computerized information systems. In *Proceedings of the IFIP WG11.3 Workshop on Database Security*.

SANDHU, R. AND JAJODIA, S. 1990. Integrity mechanisms in database management systems. In *Proceedings of the 13th NIST-NCSC National Computer Security Conference*. 526–540.

SANDHU, R. S. 1988. Transaction control expressions for separation of duties. In *Proceedings of the Fourth Annual Computer Security Applications Conference (ACSAC'88)*.

SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. 1996. Role-based access control models. *IEEE Computer 29,* 2 (February), 38–47.

SCHAAD, A., MOFFETT, J., AND JACOB, J. 2001. The role-based access control system of a European bank: A case study and discussion. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*. ACM Press, 3–9.

SIMON, T. T. AND ZURKO, M. E. 1997. Separation of duty in role-based environments. In *Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 183–194.

TAN, K., CRAMPTON, J., AND GUNTER, C. 2004. The consistency of task-based authorization constraints in workflow systems. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW)*. 155–169.

TIDSWELL, J. AND JAEGER, T. 2000. An access control model for simplifying constraint expression. In *Proceedings of ACM Conference on Computer and Communications Security*. 154–163.

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

A. SAT

A boolean literal or variable is one that can take on a value from $\{0, 1\}$. A boolean expression is one of the following: (a) a boolean variable, (b) $\neg\phi$, (c) $\phi_1 \vee \phi_2$, or (d) $\phi_1 \wedge \phi_2$,

where ϕ, ϕ_1 and ϕ_2 are boolean expressions. (c) is called a disjunction, and (d) is called a conjunction. A truth assignment to a boolean expression is the assignment of either 0 or 1 to each variable in the expression. A boolean expression is in Conjunctive Normal Form (CNF), if it can be written as $\bigwedge_{i=1}^{n} C_i$ where $n \geq 1$ and each C_i is called a clause, a clause is either a literal or a disjunction of literals, and a literal is either a boolean variable or its complement. A boolean expression is satisfiable if there exists a truth assignment to it such that it evaluates to 1. The SAT problem is the problem of determining whether an expression in CNF is satisfiable. The complement of the satisfiability problem is the validity problem: whether for any truth assignment, the expression evaluates to 1.

The 3-SAT problem is SAT with each clause has exactly 3 literals. It is well-known that SAT and MONOTONE-SAT are $\mathbf{NP}$-complete, and their complements are $\mathbf{coNP}$-complete.

Monotone 3-2-SAT is $\mathbf{NP}$-complete

MONOTONE-3-SAT is 3-SAT with each clause containing either only positive literals or only negative literals; it is known to be $\mathbf{NP}$-complete [Garey and Johnson 1979]. We use MONOTONE-3-2-SAT to denote SAT with each clause containing either 3 positive literals or 2 negative literals.

THEOREM 22. MONOTONE-3-2-SAT *is* $\mathbf{NP}$*-complete.*

PROOF. MONOTONE-3-2-SAT is clearly in $\mathbf{NP}$. We show that it is $\mathbf{NP}$-hard by reducing 3-SAT to MONOTONE-3-2-SAT.

Let $(\ell_1 \vee \ell_2 \vee \ell_3)$ be a clause. Case (1): all three literals are positive. No change needs to be made. Case (2): one is negative. Wlog, assume that ℓ_3 is negative. This clause can be equivalently represented using a positive clause $(\ell_1 \vee \ell_2 \vee w)$ and a negative clause $(\neg w \vee \ell_3)$, where w is a newly introduced propositional variable. This technique turns one literal from negative to positive by introducing a new propositional variable and a new length-2 negative clause. Case (3): two are negative. Apply the above technique twice. Case (4): three are negative. Apply the above technique three times. □