On Mutually-Exclusive Roles and Separation of Duty

Ninghui Li
ninghui@cs.purdue.edu

Ziad Bizri
zelbizri@cs.purdue.edu

Mahesh V. Tripunitara
tripunit@cerias.purdue.edu

Center for Education and Research in Information Assurance and Security
and Department of Computer Sciences
Purdue University
656 Oval Drive, West Lafayette, IN 47907

ABSTRACT

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. In Role-Based Access Control (RBAC), Statically Mutually Exclusive Role (SMER) constraints are used to enforce SSoD policies. In this paper, we pose and answer fundamental questions related to the use of SMER constraints to enforce SSoD policies. We show that directly enforcing SSoD policies is intractable (**coNP**-complete), while checking whether an RBAC state satisfies a set of SMER constraints is efficient. Also, we show that verifying whether a given set of SMER constraints enforces an SSoD policy is intractable (**coNP**-complete) and discuss why this intractability result should not lead us to conclude that SMER constraints are not an appropriate mechanism for enforcing SSoD policies. 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

Keywords

role-based access control, separation of duty, constraints, verification

1. INTRODUCTION

Separation of Duty (SoD) is widely considered to be a fundamental principle in computer security [2, 3, 17]. In its simplest form, the principle states that if a sensitive task is comprised of two steps, then different users should each 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.

Consider the following example of buying and paying for goods, taken from [3]. The steps to perform 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 the 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) at least three users' cooperation 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).

An SoD policy may be enforced either statically or dynamically. In dynamic enforcement, 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 [18, 15]. (Nash and Poland [15] 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 a particular step in a task instance; however, the user cannot also perform other steps in that instance.

In static enforcement, Static SoD (SSoD) policies are specified. 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 to have 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 a user is not participating in any active task instance while

CCS'04, October 25-29, 2004, Washington, DC, USA.

being assigned a permission, 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) [8, 9, 19]. Ferraiolo et al. [8] state, "one of RBAC's great advantages is that SoD rules can be implemented in a natural and efficient way." A purpose of this paper is to examine this statement in detail. 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 mutual exclusion constraint is Statically Mutually Exclusive Roles (SMER). An example of a SMER constraint is "no user is allowed to be a member of r_1 and r_2 simultaneously". More generally, a SMER constraint requires that no user is a member of t or more roles in a set of m roles $\{r_1, r_2, \cdots, r_m\}$. SMER constraints are part of most RBAC models, including the RBAC96 models by Sandhu et al. [19] and the proposed NIST standard for RBAC [9]. Literature in RBAC also studies dynamic mutually exclusive role (DMER) constraints. With such a constraint, a user is prevented from activating mutually exclusive roles simultaneously in a session. SMER and DMER constraints are the only types of constraints included in the proposed NIST standard for RBAC [9]. The rationale provided in that work is that such constraints are the only ones prevalent in commercial RBAC products.

As we discuss in Section 2, DMER constraints are not suitable for enforcing SoD policies, either statically or dynamically. On the other hand, SMER constraints are suitable for enforcing SoD policies statically. In this paper, we examine the use of SMER constraints to enforce SoD policies.

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 user is allowed to have.

In the literature, this distinction between objectives and mechanisms 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.

When we make a clear distinction between objectives (SSoD policies) and mechanisms (SMER constraints), several interesting problems arise. For example, the *verification* problem is whether a set of SMER constraints indeed enforces 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, surprisingly these problems have not been examined in the literature as such, 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 for the verification and generation problems.
- We show that directly enforcing SSoD policies in RBAC is intractable (**coNP**-complete), while enforcing SMER constraints is efficient.
- We show that the verification problem is intractable (**coNP**-complete), even for a basic subcase of the problem, but reduces naturally to the satisfiability problem (SAT) [5], for which there exist algorithms that have been proven to work well in practice [5]. 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 SMER constraints to enforce SoD 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 Section 5, we study the generation problem. We conclude with Section 6. Owing to space limitations, some proofs are not included in this paper; they are available in [14].

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 [17] 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 [3] identified SoD along with well-formed transactions as two major mechanisms of fraud and error control. 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.

Sandhu [18] presented a history-based mechanism for dynamically enforcing SoD policies. Nash and Poland [15] emphasized 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.

In one of the earliest paper on RBAC, Ferraiolo and Kuhn [6] used the terms Static and Dynamic SoD to refer to static and dynamic enforcement of SoD. In a subsequent paper, Ferraiolo et al. [7] 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. We believe that the terminology in [7] is confusing as it blurs the distinction between objectives and mechanisms. The same terminology is later used by several authors [1, 4, 11, 12, 20] and is adopted in the NIST proposed standard for RBAC [9].

DMER constraints are introduced in [7] under the name DSoD constraints. A DMER constraint prevents a user from simulta-

k-n SSoD policy	k-out-of-n Static Separation of Duty policy	Definition 1 in Section 3.1
t-m SMER constraint	t-out-of-m Static Mutually Exclusive Roles	Definition 4 in Section 3.2
k-n RSSoD requirement	k-out-of-n Role-based Static Separation of Duty	Definition 10 in Section 5.2
the SC-SSoD problem	the Safety Checking problem for SSoD policies	Definition 3 in Section 3.1
the SC-SMER problem	the Satisfaction Checking problem for SMER constraints	Definition 5 in Section 3.2
the EV problem	the Enforcement Verification problem	Definition 6 in Section 4
the CEV problem	the Canonical Enforcement Verification problem	Definition 8 in Section 4.1

Table 1: List of acronyms used in the paper, what they stand for, and where they are defined.

neously 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 [7]. However, as we now discuss, DMER constraints do not seem to enforce SoD policies, because they do not prevents a user from activating mutually exclusive roles across multiple sessions. In RBAC, each session has only one user. Thus, a sensitive task cannot be finished in one session; several sessions are required. Consider the example discussed in Section 1. Suppose that the permission to place an order and the permission to issue payment are assigned to two different roles that are specified to be mutually exclusive in a DMER constraint. Bob can start a session, activate the role having the order permission, create an order, end the session, start another session, activate the role having the payment permission, and authorize a payment against the order. This violates the SoD policy.

Kuhn [13] discussed mutual exclusion of roles for separation of duty and proposes a safety condition: that no one user should possess the privilege to execute every step of a task, thereby being able to execute the task. We observe that our definition for safety in Section 3.1 is a generalization of Kuhn's definition [13]: setting k to 2 gives us Kuhn's definition. Kuhn [13] did not discuss either the verification problem or the generation problem.

Simon and Zurko [20] and Gligor et al. [11] discuss various kinds of constraints and their use in RBAC. The latter discusses also the composition of constraints. Both papers refer to the constraints considered in them as SoD policies. As we discuss in this paper, this has added to the confusion in the distinction between SoD policies and the constraint mechanisms that may be used to enforce them.

Finally, several constraints languages [1, 12, 4] have been proposed to support SoD in RBAC. These papers propose more sophisticated constraint mechanisms in RBAC; whereas we study the effectiveness of using the most basic RBAC constraint mechanism, namely SMER, to enforce SSoD policies.

3. STATIC SEPARATION OF DUTY AND MUTUALLY EXCLUSIVE ROLES

In this section, we give precise definitions for Static Separation of Duty policies, RBAC, and SMER constraints. Table 1 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

Definition 1. (*SSoD policies*) 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 $1 < k \leq n$. This policy means that there should not exist a set of fewer than k users that together have all the permissions in $\{p_1,\ldots,p_n\}$. In other words, at least k users are required to perform a task that requires all these permissions.

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 execute it. The specification of an SSoD policy involves identifying a sensitive task, the permissions needed to complete it, and the minimum number of collaborating users authorized to complete it.

We now introduce the notion of RBAC states. 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).

Definition 2. (*RBAC States*) 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 $RH \subset \mathcal{R} \times \mathcal{R}$ is a partial order among roles in $\mathcal{R}$. When $(r_1, r_2) \in RH$, we 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.

An RBAC state $\gamma = \langle UA, PA, RH\rangle$ determines the set of roles a user is a member of and the set of permissions a user possesses. Formally, γ is associated with two functions, $\mathsf{roles}_\gamma\colon \mathcal{U} \to 2^{\mathcal{R}}$ and $\mathsf{perms}_\gamma\colon \mathcal{U} \to 2^{\mathcal{P}}$, 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:

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

Definition 3. (*SSoD Safety*) We say that an RBAC state γ is *safe* with respect to an SSoD policy $\mathsf{ssod}\langle\{p_1,\ldots,p_n\},k\rangle$ if in state γ no $k-1$ users together have all the permissions in the policy. More precisely,

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

An RBAC state γ is *safe* with respect to a set E of SSoD policies if it is safe with respect to every policy in the set, and we write this as $safe_E(\gamma)$.

SC-SSoD (the Safety Checking problem for SSoD policies) is defined as follows: 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), (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 the 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 adding a new user-role assignment to UA, adding a new role-permission assignment to PA, or adding 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 being violated 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 [10, 16].

The reduction is straightforward. 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. 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 ($1 \le i \le \ell$) 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 $\mathcal{S}$. □

In the proof reduction, 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. The fact that SC-SSoD is intractable suggests that enforcing SSoD policies directly can be computationally expensive.

With the following reasoning, we observe that even if we only check whether an SSoD policy is violated when adding a new user-to-role assignment, the check can be inefficient. Given an SSoD policy $e = \mathsf{ssod}\langle\{p_1, \ldots, p_n\}, k\rangle$ and an RBAC state γ that is safe with respect to e, suppose we want to check whether the state γ' that results from adding a new user-role assignment (u, r) is safe with respect to e. Let $i = |\mathsf{perms}_{\gamma'}[u] \cap \{p_1, \ldots, p_n\}|$ be the number of permissions in e that u would have in γ', then we are left with checking whether γ' is safe with respect to a $(k-1)$-$(n-i)$ SSoD policy, which remains **coNP**-complete by the proof of Theorem 1.

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 worst-case time complexity $O(N_u(N_r + N_p))$, where N_u is the number of users in γ, N_r the number of roles, and N_p the number of permissions.

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.

Definition 4. (*SMER Constraints*) 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 $1 < t \le m$. This constraint forbids a user from being a member of t or more roles in $\{r_1, \ldots, r_m\}$.

A t-m SMER constraint is said to be *canonical* of *cardinality* t when $t = m$.

Definition 5. We say that an RBAC state γ *satisfies* a t-m SMER constraint $\mathsf{smer}\langle\{r_1, \ldots, r_m\}, t\rangle$ when

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

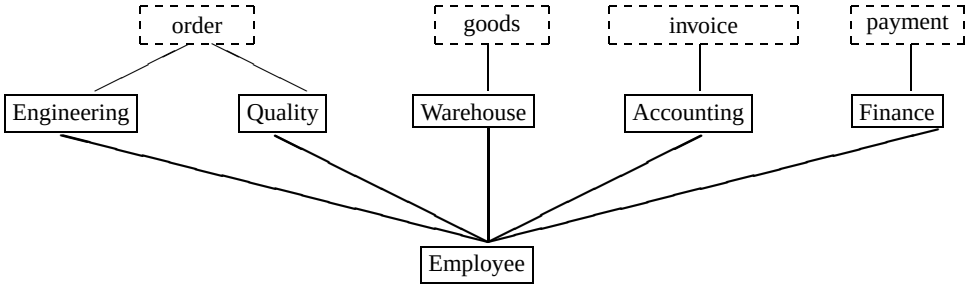
Otherwise, we say that γ *violates* the SMER constraint. An RBAC state *satisfies* a set C of SMER constraints if it satisfies every constraint in the set, and we write it as $satisfies_C(\gamma)$.

SC-SMER (the Satisfaction Checking problem for SMER constraints) is defined as follows: Given an RBAC state γ and a set C of SMER constraints, determine whether γ satisfies C.

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* **P**.

PROOF. The algorithm 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 compare 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. □



$$
\begin{aligned}
RH_1 &= \{\, (\mathsf{Engineering},\ \mathsf{Employee}),\ (\mathsf{Quality},\ \mathsf{Employee}),\ (\mathsf{Warehouse},\ \mathsf{Employee}),\\
&\quad\ (\mathsf{Accounting},\ \mathsf{Employee}),\ (\mathsf{Finance},\ \mathsf{Employee})\,\}.\\
PA_1 &= \{\, (\mathsf{Engineering},\ p_{order}),\ (\mathsf{Quality},\ p_{order}),\ (\mathsf{Warehouse},\ p_{goods}),\\
&\quad\ (\mathsf{Accounting},\ p_{invoice}),\ (\mathsf{Finance},\ p_{payment})\,\}.
\end{aligned}
$$

Figure 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.

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 E, 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 provide a justification for using SMER constraints to enforce SSoD policies. This justification is new to the best of our knowledge, as the computational complexity of SC-SSoD has not been studied in the literature.

When using SMER constraints to enforce SSoD policies, a natural question to ask is whether a set of SMER constraints is adequate to enforce a set of SSoD policies. The answer to this question depends also on the permission assignment PA and the role hierarchy RH. For instance, if all permissions in an SSoD policy are assigned to a role, then no set of SMER constraints enforces that policy.

Definition 6. Given $PA \subset \mathcal{R} \times \mathcal{P}$, $RH \subset \mathcal{R} \times \mathcal{R}$, a set E of SSoD policies, and a set C of SMER constraints. We say C *enforces* E (under PA and RH) when

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

EV (the Enforcement Verification problem) is defined as follows: Given PA, RH, a set E of SSoD policies, and a set C of SMER constraints, determine whether C enforces E (under PA and RH).

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 E_1 under PA_1 and RH_1.

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 of EV, i.e., given PA, RH, C, and E, does there exist 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, which is denoted by $\overline{\text{EV}}$. It suffices to show that $\overline{\text{EV}}$ is in **NP**. To show this, we need to show that given PA, RH, C, and E, if C does not enforce E under PA and RH, then a short (polynomial in the input size) evidence exists such that it can be verified in polynomial time.

If a set C of t-m SMER constraints does not enforce a set E of k-n SSoD policies under PA and RH, 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 7. (*SMER Equivalence*) Let C_1 and C_2 be two sets of SMER constraints. We say that C_1 and C_2 are *equivalent* when 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 a set of SSoD policies E under PA and RH if and only if C_2 enforces E under PA and RH; thus one can replace C_1 in an EV problem instance with C_2 and vice versa with no change to the enforcement.

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 8. 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 $\{e\}$.

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. Also, in the case that $t = 2$, we have a CEV instance whose size is quadratic in m.

4.2 Algorithms and complexity for CEV

It is easier to think about the complement of CEV, denoted by $\overline{\text{CEV}}$: If C does not enforce $\{\mathsf{ssod}\langle\{p_1, \cdots, p_n\}, k\rangle\}$, then there exists a user-to-role assignment for $k-1$ users such that all the SMER constraints in C are satisfied but these $k-1$ users together possess all permissions $\{p_1, \cdots, p_n\}$. It turns out that this problem is closely related to SAT, the satisfiability problem of propositional formulas in conjunctive normal form. See Appendix A for an introduction to SAT.

THEOREM 5. $\overline{\text{CEV}}$ *reduces to* SAT.

The proof of this theorem is in [14].

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.

We now show that CEV is **coNP**-hard by showing that a special case of it is **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 6. *Determining whether a set of* 2-2 *SMER constraints enforces a* 2-n *SSoD policy is* **coNP**-*complete.*

PROOF. It follows from Lemma 3. that this problem is in **coNP**.

We can prove that this problem is **coNP**-hard by reducing MONOTONE-3-2-SAT to the complement of this problem. MONOTONE-3-2-SAT is a special case of SAT where each clause contains either three positive literals or two negative literals. The details of the reduction and the proof that MONOTONE-3-2-SAT is **NP**-complete is in [14]. □

COROLLARY 7. EV *and* CEV *are* **coNP**-*complete.*

PROOF. Follows directly from Lemma 3 and Theorem 6. □

4.3 Efficiency of verification in practice

The fact that even the most basic form of EV is intractable is surprising. Enforcing SSoD policies directly by solving SC-SSOD is efficient for 2-n SSoD policies. These results cast 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. However, complexity class is only part of the story, and we now make some observations in favor of this approach.

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 is overall more expensive than using SMER constraints.

In the proof of Theorem 6, we use a reduction from MONOTONE-3-2-SAT to $\overline{\text{CEV}}$. In the reduction we generate a 2-n SSoD policy with n being unbounded. When a sensitive task involves only a small number of permissions, CEV can be done efficiently.

Even though CEV is intractable (**coNP**-complete), it means only that there exist difficult problem instances that take exponential amount of time in the worst case using existing algorithms. SAT has been studied extensively for several decades (see e.g. [5]). Many clever algorithms exist that can answer most instances efficiently. Problems in many fields, including databases, planning, computer-aided design, machine vision and automated reasoning, are reduced to SAT and solved using SAT algorithms. This often results in better performance than solving those problems directly. The fact that CEV naturally reduces to SAT means that one can benefit from the extensive research on SAT to provide practical enforcement checking.

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.) Given that most RBAC systems have many more users than roles, enforcement verification is likely to be more efficient in practice.

Finally, although checking whether an arbitrary set of SMER constraints enforces a set of SSoD policies may be expensive, SMER constraints may be generated from a set of SSoD policies and need not be verified. We discuss the generation of contraints in the following section.

5. GENERATING SMER CONSTRAINTS

Section 4 considers the problem of verifying that SMER constraints in RBAC enforce the desired SSoD policies. In this section we study the problem of generating a set of SMER constraints that are adequate for enforcing SSoD policies. We examine the following questions: How do we define a notion of precision in enforcing SSoD policies, as there are often multiple sets of constraints that enforce the same set of SSoD policies? How do we compare the "degree of restriction" of different sets of SMER constraints? What kinds of SMER constraints are needed in expressing SSoD policies, e.g., do 3-3 SMER constraints add additional expressive power over 2-2 SMER constraints?

5.1 Enforceability of SSoD policies

Definition 9. (*Enforceable SSoD configurations*) We define *an SSoD configuration* to be a 3-tuple $\langle PA, RH, E\rangle$, where E is a set of SSoD policies. An SSoD configuration is *enforceable* if there exists a set C of SMER constraints such that C enforces E under PA and RH.

LEMMA 8. *An SSoD configuration $\langle PA, RH, E\rangle$ is* ***not*** *enforceable if and only if there exists an SSoD policy $\mathsf{ssod}\langle\{p_1, \cdots, p_n\}, k\rangle$ in E such that $k-1$ roles together have all the permissions in $\{p_1, \cdots, p_n\}$.*

PROOF. If there exists such an SSoD policy, then no matter what set of SMER constraints we use, one can always assign $k-1$ different users to the $k-1$ roles without violating any SMER constraint, resulting in an unsafe state. On the other hand, if there does not exist such an SSoD policy, one can use 2-2 SMER constraints to declare every pair of roles in PA and RH to be mutually exclusive, this forbids any user from being assigned to two roles. Clearly, any state satisfying these constraints is safe. □

THEOREM 9. *Determining whether an SSoD configuration is enforceable is* $\mathbf{coNP}$*-complete.*

The proof can be found in [14]; it is very similar to that of Theorem 1.

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

5.2 RSSoD requirements

As SMER constraints are about role memberships and SSoD policies are about permissions, the first step in the generation of such constraints is to translate a policy on permissions to requirements on roles, using information in PA and RH. We now define such role-level SSoD requirements.

Definition 10. (*RSSoD requirements*) 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 $1 < k \leq n$. The meaning is that there should not exist a set of fewer than k users that together have memberships in all the n roles in the requirement. We also say k users are required to *cover* the set of n roles.

We say that an RBAC state γ is *safe* with respect to the above RSSoD requirement when

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

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

Given an SSoD configuration $\langle PA, RH, E\rangle$, we say that it is *equivalent* to a set D of RSSoD requirements if

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

where $\Leftrightarrow$ means logical equivalence.

Example 3. The SSoD configuration given in Figure 1 is equivalent to the following set of RSSoD requirements.

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

In [14] we discuss the generation of RSSoD requirements that are equivalent to SSoD configurations. A special case is when we are given an SSoD configuration $\langle PA, RH, \{e = \mathsf{ssod}\langle\{p_1, \ldots, p_n\}, k\rangle\}\rangle$, and each permission p_i is assigned to exactly one role r_i in PA and RH. Then the configuration is equivalent to the singleton set of RSSoD requirement $\{d = \mathsf{ssod}\langle\{r_1, \ldots, r_n\}, k\rangle\}$.

In the rest of this section, we discuss the generation of a set of SMER constraints to enforce one RSSoD requirement.

5.3 Precise enforcement of RSSoD requirements

From the proof of Lemma 8, it is clear that any enforceable SSoD configuration can be enforced using only 2-2 SMER constraints. This shows the power of 2-2 SMER constraints: they are sufficient to enforce any enforceable SSoD policy. However this might be at a great cost in terms of flexibility.

Ideally, one would like to generate SMER constraints that "precisely capture" the restrictions inherent to the RSSoD requirements. We now formalize this.

Definition 11. Let D be a set of RSSoD requirements and C be a set of SMER constraints, we say that C *enforces* D when

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

We say that C *is necessary to enforce* D when

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

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 C enforces D and is necessary to enforce D. Sometimes we abuse the terminology slightly and say a constraint c enforces an RSSoD requirement d.

We now give two cases where precise enforcement is possible.

LEMMA 10. *Given a k-k RSSoD requirement* $d = \mathsf{rssod}\langle\{r_1, \cdots, r_k\}, k\rangle$, *the constraint* $c = \mathsf{smer}\langle\{r_1, \cdots, r_k\}, 2\rangle$ *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 clearly 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 11. *Given a 2-n RSSoD requirement* $d = \mathsf{rssod}\langle\{r_1, \cdots, r_n\}, 2\rangle$, *the constraint* $c = \mathsf{smer}\langle\{r_1, \cdots, r_n\}, n\rangle$ *precisely enforces the configuration.*

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 e. If no user is a member of all n roles from the set, then clearly, 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 $safe_{\{d\}}(\gamma)$ must be false. □

In fact, as we prove in Lemma 19 (in Section 5.5 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 10 and 11 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.

5.4 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 to 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. From Lemma 8, 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 12. *There exist RSSoD requirements that cannot be enforced without using* 2-n *SMER constraints.*

PROOF. A t-t RSSoD requirement 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 2-2 SMER constraints are sufficient to enforce all enforceable SSoD configurations, other constraints are needed to enforce some SSoD configurations more precisely.

LEMMA 13. *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 a member of all n roles) precisely enforces the configuration, as was shown in lemma 10.

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 member 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.

5.5 "Good" sets of SMER constraints and their generation

As we show in this section (Lemma 19), SSoD policies cannot always be precisely enforced. Thus it is desirable to compare different sets of SMER constraints and determine which set "more precisely" enforces a set of SSoD policies. In this section, we first discuss the notion of precise enforcement. We then present an algorithm to generate singleton sets of SMER constraints that are as precise as possible.

Definition 12. 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 $C_1 \trianglerighteq C_2$) if

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

The $\trianglerighteq$ relation among all sets of SMER constraints is a partial order. When $C_1 \trianglerighteq C_2$ but not $C_2 \trianglerighteq C_1$, we say that C_1 is *more restrictive than* C_2 (denoted by $C_1 \triangleright C_2$). By definition, C_1 and C_2 are *equivalent* (Definition 7) if and only if $C_1 \trianglerighteq C_2$ and $C_2 \trianglerighteq C_1$.

When neither $C_1 \trianglerighteq C_2$ nor $C_2 \trianglerighteq C_1$, 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) $C \triangleright C'$; (2) $C' \triangleright C$ (3); C and C' are equivalent; and (4) C and C' are incomparable. In case (1), C' is preferable to C for enforcing D as it is less restrictive (and thus more precise). Similarly, in case (2), C is preferable to C'. In case (3), either C or C' can be used; the choice does not matter. In case (4), the decision to choose C over C' (or C' over C) depends on considerations other than SSoD policies.

Definition 13. Given a set D of RSSoD requirements, we say that a set C of SMER constraints is *minimal* for enforcing D if C enforces D and there does not exist a different set C' of SMER constraints such that C' also enforces D and $C \rhd C'$ (C is more restrictive than C').

Our approach to dealing with the generation problem is to generate all the sets of SMER constraints that are minimal for enforcing D (for any such set, no other set is more preferable than it) and leave the decision to choose which one to use to the system administrator.

LEMMA 14. *If a set C of SMER constraints precisely enforces a set D of RSSoD requirement, then for any C' that also enforces D, $C' \trianglerighteq C$, i.e., C' is at least as restrictive as C.*

PROOF. We need to show that: (a) for every RBAC state γ, $satisfies_{C'}(\gamma)$ implies $satisfies_{C}(\gamma)$. We show that this is equivalent to proving (b) for every RBAC state γ' such that $live_D(\gamma')$, $satisfies_{C'}(\gamma')$ implies $satisfies_{C}(\gamma')$. (a) clearly implies the (b). 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 now 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 resulting state γ_2. Clearly, $live_D(\gamma_2)$ is true. Furthermore, $satisfies_{C'}(\gamma)$ if and only if $satisfies_{C'}(\gamma_2)$, and $satisfies_{C}(\gamma)$ if and only if $satisfies_{C}(\gamma_2)$. Therefore, $satisfies_{C'}(\gamma_2)$ is true but $satisfies_{C}(\gamma_2)$ is not. This is in contradiction to (b) being true.

If C' enforces D, then for every γ such that $live_D(\gamma)$ is true, $satisfies_{C'}(\gamma)$ implies $safe_D(\gamma)$, which further implies $satisfies_{C}(\gamma)$. Thus C' is at least as restrictive as C. □

LEMMA 15. *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 14 that $C' \trianglerighteq C$, thus it cannot be $C \rhd C'$ (which implies that $\neg(C' \trianglerighteq C)$).

Given any C_1 that is also minimal for enforcing D, it follows from Lemma 14 that $C_1 \trianglerighteq C$. By Definition 13, it cannot be that $C_1 \rhd C$; thus C_1 and C must be equivalent. □

LEMMA 16. *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 exist a set C that precisely enforces D, then from Lemma 15, C is equivalent to C_1 and to C_2. This contradicts the fact that C_1 and C_2 are incomparable. □

We now present an algorithm that generates all singleton sets of SMER constraints that are minimal for enforcing one RSSoD requirement.

The SMER-Gen Algorithm
Input: RSSoD requirement rssod$\langle R, k\rangle$
Output: a set S of minimal SMER constraints

```
1 let n = |R|, S = ∅
2 if k = 2
3   output smer⟨R, n⟩
4 else
5   for all j from 2 to ⌊(n−1)/(k−1)⌋ + 1
6     let m = (k − 1)(j − 1) + 1
7     for each size-m subset R′ of R
8       output smer⟨R′, j⟩
9 end
```

Example 4. The above algorithm, with rssod $\langle\{\mathsf{Engineering}, \mathsf{Warehouse}, \mathsf{Accounting}, \mathsf{Finance}\}, 3\rangle$ as input, generates the sets of SMER constraints $\{c_4\}$, $\{c_5\}$, $\{c_6\}$, and $\{c_7\}$ where

c_4 = smer$\langle\{\mathsf{Warehouse}, \mathsf{Accounting}, \mathsf{Finance}\}, 2\rangle$
c_5 = smer$\langle\{\mathsf{Engineering}, \mathsf{Accounting}, \mathsf{Finance}\}, 2\rangle$
c_6 = smer$\langle\{\mathsf{Engineering}, \mathsf{Warehouse}, \mathsf{Finance}\}, 2\rangle$
c_7 = smer$\langle\{\mathsf{Engineering}, \mathsf{Warehouse}, \mathsf{Accounting}\}, 2\rangle$

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 17. *Given an RSSoD requirement d, each SMER constraint generated by SMER-Gen(d) is minimal for enforcing d.*

LEMMA 18. *Given an RSSoD requirement d, every SMER constraint that is minimal for enforcing d is generated by SMER-Gen(d).*

The proofs for these two lemmas are in [14]. Lemmas 16 and 17 enable us to prove the following Lemma.

LEMMA 19. *Given a k-n RSSoD requirement where $2 < k < n$, there exists no set of SMER constraints that precisely enforces it.*

PROOF. 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 17) and therefore there exists no set of SMER constraints that precisely enforces the RSSoD requirement (Lemma 16). The details of the proof are in [14]. □

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.

6. 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 (**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 (**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 [5]. 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 [19]. It would be interesting to examine the use of SMER constraints together with these constraints to enforce SSoD policies.

Acknowledgement

Portions of this work were supported by NSF ITR, Purdue Research Foundation, and sponsors of CERIAS. We thank Trent Jaeger for helpful discussions. We thank the anonymous reviewers for their helpful comments. We thank also Elisa Bertino, Ji-Won Byun, Jiangtao Li and Klorida Miraj at CERIAS for reading a draft of the paper and making suggestions that have improved the paper's presentation.

7. REFERENCES

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

[2] M. Bishop. *Computer Security — Art and Science*. Addison-Wesley, 2003.

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

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

[5] D. Du, J. Gu, and P. M. Pardalos, editors. *Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. AMS Press, 1997.

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

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

[8] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Artech House, Apr. 2003.

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

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

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

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

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

[14] N. Li, Z. Bizri, and M. V. Tripunitara. On mutually-exclusive roles and separation of duty. Technical Report CERIAS-TR-2004-21, Center for Education and Research in Information Assurance and Security, Purdue University, June 2004.

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

[16] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley Longman, 1994.

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

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

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

[20] T. T. Simon and M. E. Zurko. Separation of duty in role-based environments. In *Proceedings of The 10th Computer Security Foundations Workshop*, pages 183–194. IEEE Computer Society Press, June 1997.

APPENDIX

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.