PAPER *Special Section on Frontiers of Information Network Science*

Static Enforcement of Static Separation-of-Duty Policies in Usage Control Authorization Models

Jianfeng LU[†*], ***Nonmember***, **Ruixuan LI**[††], ***Member***, **Jinwei HU**[†††], ***and*** **Dewu XU**[†], ***Nonmembers***

SUMMARY Separation-of-Duty (SoD) is a fundamental security principle for prevention of fraud and errors in computer security. It has been studied extensively in traditional access control models. However, the research of SoD policy in the recently proposed usage control (UCON) model has not been well studied. This paper formulates and studies the fundamental problem of static enforcement of static SoD (SSoD) policies in the context of UCON_A, a sub-model of UCON only considering authorizations. Firstly, we define a set-based specification of SSoD policies, and the safety checking problem for SSoD in UCON_A. Secondly, we study the problem of determining whether an SSoD policy is enforceable. Thirdly, we show that it is intractable (coNP-complete) to direct statically enforce SSoD policies in UCON_A, while checking whether a UCON_A state satisfies a set of static mutually exclusive attribute (SMEA) constraints is efficient, which provides a justification for using SMEA constraints to enforce SSoD policies. Finally, we introduce a indirect static enforcement for SSoD policies in UCON_A. We show how to generate the least restrictive SMEA constraints for enforcing SSoD policies in UCON_A, by using the attribute-level SSoD requirement as an intermediate step. The results are fundamental to understanding SSoD policies in UCON.

key words: *Separation-of-Duty, usage control, constraint, static mutually exclusive attribute.*

1. Introduction

Separation-of-Duty (SoD) is widely considered to be a fundamental principle for prevention of fraud and errors in computer security, and widely applied in business, industry, and government [1], [2]. The concept of SoD has long existed before the information age, sometimes under the name "the two-man rule". One of the best known requirements for SoD is embodied in the Chinese Wall model [3], in which access to documents that could result in a commercial conflict of interest is strictly controlled. The literature has long recognized that there are two major types of SoD policies: dynamic SoD (DSoD) and static SoD (SSoD)[4]. A simple way to distinguish between DSoD and SSoD is to consider the time at which the permission are authorized. DSoD typically constrains the activation of permissions at run time. SSoD typically constrains the assignment of permissions to users. DSoD is a weaker form of SSoD [5], since, for example, it may allow a user to be authorized for both permissions p_1 and p_2, but not allow the user to hold these permissions in a single session. SSoD is an important variation of SoD, an 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. It provides the capability to address potential conflict-of-interest issues when a permission is assigned to a user.

SoD has been studied extensively in the traditional access control models; it has been recognized that "one of role-based access control (RBAC)'s great advantages is that SoD rules can be implemented in a natural and efficient way" [6]. Recently, usage control [7] was proposed as a general and comprehensive model to extend the underlying mechanism of traditional access control models. UCON is a new access control model that covers traditional access controls such as mandatory, discretionary, role-based access control, and other models like digital rights management and other modern access controls [8]. It has been considered as the next generation access control model with distinguishing properties of decision continuity and attribute mutability. However, as a related and fundamental problem, the research of SoD policy in usage control (UCON) model [7] has not been fully explored. Since authorization decisions in UCON are not only checked and made before the access, but may be repeatedly checked during the access and may be revoked if some policies are not satisfied, according to the changes of the subject or object attributes, or environmental conditions, the enforcement of SoD policies in UCON is of course more difficult than in RBAC.

This paper focuses on the SSoD policy in UCON_A [7], [8], a sub-model of UCON only considering authorizations for the following reasons. Firstly, SSoD policies can be enforced statically, which ensures that each access control state that can be reached is safe with respect to the SSoD policies for the task. Furthermore, static enforcement can be achieved either directly or indirectly. In direct static enforcement, before making

Manuscript received September 7, 2011.
Manuscript revised December 31, 2011.

[†]The authors are with School of Mathematics-Physical and Information Engineering, Zhejiang Normal University, Jinhua 321004, P. R.China.
[††]The author is with school of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, P. R.China.
[†††]The author is with Department of Computer Science, College of Engineering, Qatar University, Doha, Qatar.
[†]E-mail: lujianfeng@zjnu.edu.cn

DOI: 10.1587/transcom.E0.B.1

changes to the access control state, one checks that the resulting state is safe and makes the change only when it is safe. In indirect static enforcement, one specifies constraints so that any access control state satisfying the constraints is safe and thus only need to check whether a resulting state satisfies the constraints during state changes. We argue that the study of indirect static enforcement of SSoD policies should be given highest priority than the direct static one, since we show that direct static enforcement is computationally intractable (coNP-complete). On the other hand, it is efficient to check whether an UCON_A state satisfies a set of static mutually exclusive attribute (SMEA) constraints, which provides a justification for using SMEA constraints to enforce SSoD policies. Secondly, The family of UCON_ABC models for UCON, which integrate Authorizations (UCON_A), oBligations (UCON_B), and Conditions (UCON_C). In UCON_A, the control decision of an access is determined by one or more predicates built from the attributes of the subject and the object. Since an authorization decision is determined by subject's and object's attributes, and these attribute values can be updated as side-effects of the authorization, the study of SSoD policies in UCON_A is more pressing than that in UCON_B and UCON_C. In authorization models, usage control decisions can be checked and determined before or during a usage process, which are called *preA* (pre-authorizations) and *onA* (ongoing authorizations), respectively. For UCON_onA models, the system state changes nondeterministically, depending on concurrent accesses and reasons for attribute updates. We leave the research of the SSoD policies in UCON_onA models for future work. For the sake of simplicity in this paper we refer UCON_A as UCON_preA models. The research of indirect static enforcement of SSoD policies in UCON_A is important for emerging applications as usage control scenarios, and it can also increase UCON's strengths in that it enables the use of constraints to support SSoD policies.

Our contributions in this paper are as follows:

- We define a set-based specification of SSoD policies and the safety checking problem for SSoD policies in the context of UCON_A systems.
- We study the problem of determining whether an SSoD policy is enforceable, and show that whether an SSoD policy is enforceable is determined by the number of ancestor attribute-sets for it.
- We show that direct statically enforcing SSoD policies in UCON_A is intractable (coNP-complete) in general, which means that there exist difficult problem instances that take exponential time in the worst case.
- We show that checking whether a UCON_A state satisfies a set of static mutually exclusive attribute (SMEA) constraints is efficient, which provides a justification for using SMEA constraints to enforce SSoD policies. Therefore, we generate SMEA constraints that are as accurate as possible for indirect static enforcing SSoD policies in UCON_A, by using the attribute-level SSoD requirement as an intermediate step.

The rest of this paper is organized as follows. Section 2 contains a brief introduction of UCON_A. Section 3 gives the specification of SSoD policies, as well as the safety checking problem for SSoD policies in the context of UCON_A models. Section 4 studies the problems of determining whether an SSoD policy is enforceable in the context of UCON_A. Section 5 studies the computational complexities for direct statically enforcing SSoD policies in UCON_A. Section 6 shows how to use SMEA constraints for indirect statically enforcing SSoD policies. Section 7 describes related works, and section 8 concludes this paper and presents further directions of research.

2. A Brief Introduction of UCON_A

A UCON system consists of six components: subjects and their attributes, objects and their attributes, generic rights, authorizations, obligations, and conditions, where authorizations, obligations and conditions are the components of usage control decisions [7], [8]. Where authorizations are predicates based on subject/object attributes, obligations are actions that are performed by subjects or system, and conditions are system and environmental restrictions. The most important properties that distinguish UCON from traditional access control models are decision continuity and attribute mutability. Where decision continuity components are checked and enforced in the before-usage and ongoing-usage phases, and attribute mutability means that one ore more subject or object attribute values can be updated as the results of an access.

UCON_A is a sub-model of UCON only considering authorizations. The authorization decision in UCON_A is determined by subject/object attributes and system attributes, and these attribute values can be updated as side-effects of authorization. A permission is a triple (s, o, r), where s, o, r are a subject, object, and right, respectively. The permission enables the access of a subject s to an object o in a particular mode r, which is enabled by an authorization rule in an authorization policy. A UCON_A state is specified by attribute-value assignments for each object in the system. In each system state, a predicate is evaluated by using the values of the attributes in the state. A formal representation of a UCON_A system can be defined with the basic components that we have introduced. The UCON_A scheme affects the system in two ways. First, a set of satisfied pre-authorization policies can authorize a permission so that a subject can access an object with this particular permission. Second, the predicates may authorize

the system to move to a new state with a sequence of actions. In this way, the state of a UCON_A system focuses on the interactions between these two aspects. An assignment of an attribute maps its attribute name to a value in its domain, denoted as $o.a = v$, where $v \in dom(a) \cup null$, $dom(a)$ denotes the domain of the attribute a. The set of assignments for all objects' attributes collectively constitute a state of the system.

Definition 1. *A* UCON_A *state* ε *is a pair* (O, θ)*, where* O *is a set of objects, and* $\theta : O \times ATT \rightarrow dom(ATT) \cup \{null\}$ *is a function that assigns a value or null to each attribute of each subject or object.*

A subject is a unit of access control in UCON_A, and a user may have multiple subjects (or sessions) with different permissions active at the same time. We assume that there is only one access generated from a single user. A UCON_A state $\varepsilon = (S, \theta)$ directly determines subject's attributes, and indirectly determines the subject's permissions. We use $ATT(s)$ and $ATT(o)$ to denote the subject's attributes and object's attributes. UCON_A utilizes $ATT(s)$, $ATT(o)$, and permission for usage decision making. We do not consider the internal structure of permission. In this way, UCON_A only examines usage requests using $ATT(s)$ and permission then decides if the access request is allowed or denied.

3. The Specification of SSoD Policies

SSoD is not a complicated concept; it is easy to motivate and understand intuitively. Several definitions of SSoD have been given in the literature, research papers on SSoD policies in computer systems regularly describe constraints that are defined in terms of users and roles. We now give a formal basis for this principle in computer security systems. The definition of SSoD policies are based on the following requirements.

(1)An SSoD policy must be a high-level requirement. Clark et al.[9] identified an SSoD policy as a high-level mechanism that is "at the heart of fraud and error control". It states a high-level requirement about the task without the need to refer to individual steps in the task. Consequently, an SSoD policy states an overall requirement that must be satisfied by any set of users that together complete a task, rather than restricting which users are allowed to carry out the individual steps. As the specification abstracts away details of how a task is implemented, an SSoD policy is likely to be closer to organizational policy guidelines; in other words, the SSoD policies must be task-level policies rather than step-level policies.

(2)An SSoD policy must be expressed in terms of restrictions on permissions. In the literature, SSoD is usually defined not in terms of permissions. As in the ANSI RBAC standard [10], static mutually exclusive role (SMER) constraints are called SSoD constraints, and dynamic mutually exclusive role (DMER) constraints are called DSoD constraints. The distinction between SSoD policies as objectives and SMER constraints as a mechanism is not clearly. One danger is that the SMER or DMER constraints may be specified without a clear specification of what objectives they intend to meet; consequently, it is unclear whether the higher-level objectives are met by the constraints or not. Another danger is that even though when SMER or DMER constraints are specified, and there exists 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 [11]. However, these two dangers will be relieved while SSoD policies are expressed in terms of restrictions on permissions.

(3)An SSoD policy must capture restrictions on the user set involved in the task. In general, the user set is the set of all possible users in the system; but in practice, the number of users in any organization is bounded. This makes SSoD policies harder to be satisfied in a given access control state [12]. Therefore, the definition of SSoD policies in this paper considers the total number of available users as a limiting factor.

An 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. We now formally define SSoD policies as follows.

Definition 2. *An SSoD policy is expressed as*

$$ssod\,\langle\{p_1, \cdots, p_m\}, \{u_1, \cdots, u_n\}, k\rangle$$

where each p_i *is a permission needed to complete a sensitive task, each* u_j *is a user authorized to complete the task,* m*,* n*, and* k *are integers, such that* $2 \leq k \leq min(m, n)$*, min returns the smaller value of the two.*

Intuitively, the policy $ssod\,\langle P, U, k\rangle$ means that there should not exist a set of fewer than k users from $\{u_1, \cdots, u_n\}$ required to perform a task that together have all the permissions in $\{p_1, \cdots, p_m\}$. Obviously, the SSoD policy precludes any group of users from possessing too many permissions, but a UCON_A state directly determines subject's attributes, and indirectly determines the subject's permissions. Due to their opposite objectives, an SSoD policy can not be satisfied by a given UCON_A state.

Definition 3. *We say that a* UCON_A *state* ε *is safe with respect to an SSoD policy* $e = ssod\,\langle P, U, k\rangle$*, which we denote by* $safe_e(\varepsilon)$*, if and only if in state* ε *no k-1 users from* U *together have all the permissions in* P*. Formally,*

$$\forall\{u_1, \cdots, u_{k-1}\} \subseteq U \left(\bigcup_{i=1}^{k-1} auth\_p_\varepsilon(u_i')\right) \not\supseteq P$$

where $auth\_p_\varepsilon[u] = \{p|allowed(u,p) \Rightarrow preA(ATT(u),p)\}$, $ATT(u)$ *denotes the user's attributes,* $allowed(u,p)$ *indicates that user* u *is assigned the permission* p*, and* $preA(A,p)$ *is the pre-authorizations in* UCON_A*, any user covers all the attributes in the attribute set* A*, it can be assigned the permission* p*.*

Definition 4. *Given a* UCON_A *state* ε *and an SSoD policy* e*, determine whether* $safe_e(\varepsilon)$ *is true is called the safety checking problem for SSoD (SC-SSoD).*

Observe that if no k-1 users together have all the permissions in an SSoD policy, then no set of fewer than k users together have all the permissions. If the security administrator wants to specify an SSoD policy, he should first identify a sensitive task, and then identify the permissions in P that are needed to carry out a sensitive task, the constraint set of user set U, and determine the minimum number k of collaborating users should be authorized to complete it. A UCON_A state ε is safe with respect to a set E of SSoD policies, which we denote by $safe_E(\varepsilon)$, if and only if ε is safe with respect to every SSoD policy $e \in E$.

4. Enforceability of SSoD Policies

In practice, not all SSoD policies are enforceable in a UCON_A system. For example, given an SSoD policy e and a UCON_A state ε, e may be incompatible with ε that some attributes in the system useless. To address this, we use a running example to illustrate the concept of enforceability as follows, and present a necessary and sufficient condition for determining whether an SSoD policy is enforceable in a UCON_A state.

Example 1. *Suppose that each user in an organization has the same set of attribute names* $ATT = \{Role, Identity, Digital\_money\}$*, where the Role's value is a system role name,* $dom(Role) = \{engineer, programmer, supervisor\}$*. The Identity denotes the identity of a user,* $dom(Identity) = \{student, teacher\}$*, the Digital_money is a numerical value,* $dom(Digital\_money) = \{v|0 \le v \le 1000\}$*. Assume that the system administrator declares that each user requesting the permission* p_1 *or* p_2 *must comply with the following preA predicates:*

- *allowed(u,*p_1*)⇒ preA(ATT(u),*p_1*), where ATT(u)= {{engineer,student},{50}}*
- *allowed(u,*p_2*)⇒ preA(ATT(u),*p_2*), where ATT(u)= {{programmer,student},{150}}*

For any user u*, if* u *has the engineer role, he is a student, and his Digital_money is no less than 50, then* p_1 *can be assigned to* u*. Similarly, if* u *has the programmer role, he is a student, and his Digital_money is no less than 150, then* p_2 *can be assigned to* u*.*

We assume that the permissions in $\{p_1, p_2\}$ are needed to carry out a sensitive task and a policy guarantees that at least two users are needed to successfully complete it. One may declare p_1 and p_2 to be mutually exclusive that no user from the user set $\{Alice, Bob, Carl\}$ is allowed to be a member of both, he may specify an SSoD policy $e = ssod\,\langle\{p_1, p_2\}, \{Alice, Bob, Carl\}, 2\rangle$. Let's suppose that $ATT(Alice) = \{\{supervisor, student\}, \{200\}\}$, where *supervisor* is a senior role to both *engineer* and *programmer*. Obviously, $safe_e(\varepsilon)$ is false, because *Alice* can be a member of both p_1 and p_2 in that $ATT(Alice)$ according to $allowed(u, p_1)$ and $allowed(u, p_2)$. In order to address this, one may forbid *Alice* from having the attribute-set $\{\{supervisor, student\}, \{200\}\}$. This is undesirable; if an attribute value cannot be assigned to a user, then it should not be included in the domain of the user attribute. This is the key problem in this section. For this, we need to determine whether a given SSoD policy can be enforced in a UCON_A state, which is based on the comparison of attribute sets.

Definition 5. (I, M) *is an attribute set, where* I *is the set of immutable attributes,* M *is the set of mutable attributes. We say that* (I_j, M_j) *is senior to* (I_i, M_i)*, denoted by* $(I_i, M_i) \preceq (I_j, M_j)$*, if and only if for each attribute* $a \in I_i$*, there exists an attribute* $a' \in I_j$ *such that* a' *is senior to* a*, and denoted by* $a \preceq a'$*; and for each attribute* $b \in M_i$ *there exists an attribute* $b' \in M_j$ *such that* b' *is greater than* b*, and denoted by* $b \preceq b'$*.*

Mutable attributes are modified by the system automatically that do not require any administrative actions for update. Immutable attributes cannot be changed by the subject's activity, only administrative actions can change it. Obviously, $\preceq$ associates the user attribute-sets be reflexive, transitive, and anti-symmetric, thus these associations form a combined hierarchy that is partially ordered. Continuing from Example 1, consider the following set of user attributes: $(I_1, M_1) = \{\{engineer, student\}, \{50\}\}$, $(I_2, M_2) = \{\{programmer, student\}, \{150\}\}$, and $(I_3, M_3) = \{\{supervisor, student\}, \{200\}\}$. AS $engineer \preceq supervisor$, $student \preceq student$, and $100 \preceq 200$, thus $(I_1, M_1) \preceq (I_3, M_3)$. Similarly, $(I_2, M_2) \preceq (I_3, M_3)$. But $(I_1, M_1) \not\preceq (I_2, M_2)$, since $engineer \not\preceq programmer$.

Definition 6. (I, M) *is the threshold attribute-set of* p*, if and only if* $\forall u \in U(allowed(u, p) \Rightarrow (ATT(u) = (I, M)))$*, denoted by* $(I, M)_p$*.*

Definition 7. *Given an SSoD policy* $e = ssod\,\langle P, U, k\rangle$*,* $(I, M)_{p_i}$ *is the threshold value of each* p_i *in* $\{p_1, \cdots, p_m\}$*, and* (I_t, M_t) *is an attribute-set. We say* (I_t, M_t) *is an ancestor attribute-set for* e*, if and only if* $\forall (I, M)_{p_i}(I \preceq I_t \Rightarrow M \preceq M_t)$*.*

In Example 1, assume that $e_1 = ssod\,\langle\{p_1, p_2\}, U, 2\rangle$

is an SSoD policy, and $(I_{p_1}, M_{p_1}) = \{\{engineer, student\}, \{50\}\}$, $(I_{p_2}, M_{p_2}) = \{\{programmer, student\}, \{150\}\}$. Then (I_3, M_3) is an ancestor attribute-set for e_1, as $(I_1 \preceq I_3 \Rightarrow M_1 \preceq M_3) \wedge (I_2 \preceq I_3 \Rightarrow M_2 \preceq M_3)$.

Lemma 1. *The number of ancestor attribute-sets for an SSoD policy is equal to the least number of ancestors of immutable attributes in these attribute-sets.*

Proof. In a given SSoD policy $e = ssod\langle P, U, k\rangle$, where $P = \{p_1, \cdots, p_m\}$, $U = \{u_1, \cdots, u_n\}$, for each permission p_i in P, there exists a threshold attribute-set $(I, M)_{p_i}$ corresponding to p_i. Assume $\{I_1, \cdots, I_k\}$ is the least number of the ancestors of $\{I_{p_i}, \cdots, I_{p_m}\}$, that means there does not exist less than k immutable attribute-sets that together cover all immutable sets in $\{I_{p_i}, \cdots, I_{p_m}\}$. For each I_i in $\{I_1, \cdots, I_k\}$, assume I_i is senior to some immutable attribute-sets in $\{I^{'}_{p_1}, \cdots, I^{'}_{p_t}\} \subseteq \{I_{p_i}, \cdots, I_{p_m}\}$, Let $(I^{'}_{p_1}, M^{'}_{p_1})$, $(I^{'}_{p_2}, M^{'}_{p_2})$ and $(I^{'}_{p_t}, M^{'}_{p_t})$ be the corresponding attribute-sets. We can construct the least number of the ancestor attribute-sets (I, M) for e as follows, let $I = I_i$, and $M = max(M^{'}_{p_1}, \cdots, M^{'}_{p_t})$, where max returns the senior-most mutable attribute-set of them. It is easily to proof that (I, M) is an ancestor attribute-set for e. □

Definition 8. *We say an SSoD policy e is unenforceable in a* UCON_A *state ε, if and only if there exists an attribute set (I,M) such that for any user-attribute assignment relation ATT(u) which satisfies e under ε, $\nexists u \in U((I, M) \preceq ATT(u))$. e is enforceable in ε if and only if e is not unenforceable in ε.*

Previous observations lead to Theorem 1, which is helpful in verifying whether an SSoD policy is enforceable in a given UCON_A state. We now present a necessary and sufficient condition for an SSoD policy to be enforceable in a UCON_A state as follows.

Theorem 1. *An SSoD policy $e = ssod\langle P, U, k\rangle$ is unenforceable if and only if the number of ancestor attribute-sets for e is less than k.*

Proof. For the "if" part, we assume that if the condition in the theorem holds. Then one can construct a UCON_A state in which there are k-1 users and each of the users in U is assigned one of the k-1 ancestor attribute-set (I, M) for e. Thus these k-1 users together cover all the permissions in P, and result in an unsafe state.

For the "only if" part, we show that if the condition in the theorem does not hold, then the SSoD policy is enforceable. Consider that the number of ancestor attribute-sets is k. We can declare every pair of (I, M) to be mutually exclusive, which forbids any user to cover any two of them, thus $safe_e(\varepsilon)$ is true. □

Observe that if no k-1 ancestor attribute-sets (I, M) such that these k-1 ancestor attribute-sets together have all the permissions in $\{p_1, \cdots, p_m\}$, then the SSoD policy is enforceable. Lemma 1 shows that the number of least ancestors for attribute-sets (I, M) is determined by the immutable attributes. Therefore, we only need to check whether the number of ancestor of attribute-sets (I, v) is less than k, where v is a constant. The following cases may arise while computing the number of ancestor for attribute-sets (I, v):

- There is only one type of immutable attributes related to permissions in $\{p_1, \cdots, p_m\}$. e.g., the permissions are only related to roles.
- There are many types of immutable attribute related to permissions in $\{p_1, \cdots, p_m\}$. e.g., in Example 1, the permissions are related to both roles and identities.

We propose an algorithm to automatically compute the number of ancestors for attribute-sets (I, v) in Algorithm 1. This algorithm assumes that the immutable attribute is stored in a forest data structure. This algorithm has a time complexity of $O({N_I}^2)$, where N_I is the max number for each type of the immutable attributes in a UCON_A state. For the second case that there are many types of immutable attribute related to permissions in $\{p_1, \cdots, p_m\}$, we can decompose this case to many single types in the first case, after computing each result, the final result is to multiply each previous result.

5. Direct Statically Enforcing SSoD Policies

Given an enforceable SSoD policy set E in a UCON_A state ε, one can employ direct or indirect static enforcement of SSoD policies in ε. And in this section, we show that direct statically enforcing SSoD policies is coNP-complete in general, which means that there exist difficult problem instances that take exponential time in the worst case.

Theorem 2. SC-SSoD *is coNP-complete.*

Proof. Consider the complement of SC-SSoD, i.e., given an access control state ε and an SSoD policy e, determine if $safe_e(\varepsilon)$ is false, which is denoted by $\overline{\text{SC-SSoD}}$.

We first show that $\overline{\text{SC-SSoD}}$ is in NP. If an access control state ε is not safe with respect to an SSoD policy $e = ssod\langle\{p_1, \cdots, p_m\}, \{u_1, \cdots, u_n\}, k\rangle$, there must exist k-1 users in $\{u_1, \cdots, u_n\}$ that together have all the m permissions in $\{p_1, \cdots, p_m\}$. If one correctly guesses the k-1 users that together have all the m 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. But when verifying problem of $safe_e(\varepsilon)$, one only needs to compute the set of permissions of every $size$-(k-1) user

Algorithm 1. Computing the number of the least ancestor of attribute-sets (I, v)

Input: an attributes hierarchy A, the attribute-set I
Output: the number of least ancestor of (I, v)

```
1: initialize k = 0; S = U = T = ∅; // S, U and T are three attribute-sets
2: foreach attribute a in I do //computing the ancestor attributes for attribute-set I
3:    if a has no senior attributes then
4:       foreach attribute i in I do
5:          if a is a senior attribute to i then
6:             S = S ∪ {a}
7:          end if
8:       end foreach
9:    end if
10: end foreach
11: foreach attribute s in S do
12:    Junior(s) = ∅ // Junior(s) is an attribute-set
13:    foreach attribute i in I do
14:       if s is senior to i then
15:          Junior(s) = Junior(s) ∪ {i}
16:       end if
17:    end foreach
18: end foreach
19: foreach attribute s in S do
20:    U = U ∪ Junior(s)
21: end foreach
22: U = U − I
23: while k = 0
24:    foreach attribute s in S do
25:       k = 0
26:       if Junior(s) ⊆ U then
27:          k = 1
28:       end if
29:       if count(Junior(s)) = 1 then
30:          S = S − s
31:          U = U − {Junior(s)}
32:       else then
33:          T = T ∪ {s}
34:       end if
35:    end foreach
36:    if t has the minimal junior attributes in T then
37:       S = S − {t}
38:       U = U − {Junior(t)}
39:    end if
40: end while
41: return count(S)
```

sets in $\{u_1, \cdots, u_n\}$, and check whether it is a superset of $\{p_1, \cdots, p_m\}$. The running time for this straightforward algorithm grows polynomially in the number of users and permissions and exponentially only in k. Therefore, $\overline{\text{SC-SSoD}}$ is in NP.

We now show that $\overline{\text{SC-SSoD}}$ is NP-hard by reducing the NP-complete set covering problem [13] to it. In the set covering problem, the inputs are a finite set S, a family $F = \{S_1, \cdots, S_l\}$ of subsets of S, and a budget B. The goal is to determine whether there exist B sets in F whose union is S. This problem is NP-complete. The reduction is as follows. Given S, F and B, construct an SSoD policy e as follows: for each element in S, we create a permission for it, let k be $B + 1$ and let m be the size of S. We construct a SSoD policy $ssod\langle S, \{u_1, \cdots, u_n\}, B + 1\rangle$, and construct a UCON_A state as follows. For each different subset $S_i (1 \leq i \leq l)$ in F, create a user $u_i \in \{u_1, \cdots, u_n\}$, to which all permissions in S_i are assigned. The resulting $safe_e(\varepsilon)$ is false if and only if B sets in F cover S. Therefore, $\overline{\text{SC-SSoD}}$ is NP-hard.

□

While the verification problem of $safe_e(\varepsilon)$ is intractable in general, efficient algorithms exist when k is small or close to m. Lemma 2 and Lemma 3 show that the two cases where $k = 2$ and $k = m$ can be enforced efficiently. When $k = 2$, any user from $\{u_1, \cdots, u_n\}$ have all the permissions in $\{p_1, \cdots, p_m\}$ that resulting $safe_e(\varepsilon)$ is false, otherwise $safe_e(\varepsilon)$ is true. When $k = m$, every permissions in $\{p_1, \cdots, p_m\}$ be assigned to the users in $\{u_1, \cdots, u_n\}$, and there exists a user from $\{u_1, \cdots, u_n\}$ has any two permissions in $\{p_1, \cdots, p_m\}$, then $safe_e(\varepsilon)$ is false, otherwise $safe_e(\varepsilon)$ is true.

Lemma 2. *Given an SSoD policy $e = ssod\langle P, U, 2\rangle$, and a UCON_A state ε,$safe_e(\varepsilon)$ is true if and only if no user in U cover all permissions in P.*

Proof. The SSoD policy e requires that two users are required to cover all m permissions. For the "only if" part, if $safe_e(\varepsilon)$ is true, then no user is allowed to be authorized for all m-1 permissions in P, then at least two users are required to cover all the m-1 permissions. For the "if" part, if a user cover all permissions in P, then $safe_e(\varepsilon)$ is false. □

Lemma 3. *Given an SSoD policy $e = ssod\langle P, U, m\rangle$, and a UCON_A state ε, let every permissions in P be assigned to the users in U, $safe_e(\varepsilon)$ is true if and only if no user is a member of two permissions in P.*

Proof. The SSoD policy e requires that m users are required to cover all m permissions. For the "only if" part, if $safe_e(\varepsilon)$ is false, then m-1 users from U together have m permissions in P, there must exist a user in U who has no less than two permissions in P. For the "if" part, if there is a user in U has two permissions in P, then there must be m-2 users are required to cover the rest m-2 permissions in P, the resulting m-1 users together have all the permissions in P, thus $safe_e(\varepsilon)$ is false. □

6. Indirect Statically Enforcing SSoD Policies by SMEA Constraints

Although direct statically enforcing SSoD policies is intractable, while checking whether a UCON_A state satisfies a set of SMEA constraints is efficient, that provides a justification for using SMEA constraints to enforce SSoD policies. Therefore, we give algorithms to generate SMEA constraints to enforce SSoD policies by following two steps: the first step is to translate restrictions on permissions expressed in SSoD policies to

restrictions on attribute-set (I, M), and the second step is to generate SMEA constraints from a set of attribute-level SSoD requirements.

6.1 SMEA Constraints and ASSoD requirements

In the context of RBAC systems, constraints such as static mutually exclusive roles (SMER) are introduced to enforce SSoD policies. In the most basic form of SMER, two roles may be declared to be mutually exclusive in the sense that no user is allowed to be a member of both roles; in the general form, it forbids a user from being a member of k or more roles. Since UCON_A includes RBAC, and role is a special type of attributes in UCON_A [7]. We present a generalized form of the SMEA constraints in this paper, which is directly motivated by SMER constraints.

Definition 9. *A statically mutually exclusive attribute (SMEA) constraint is expressed as*

$$smea\,\langle\{(I_1, M_1), \cdots, (I_m, M_m)\}, \{u_1, \cdots, u_n\}, k\rangle$$

where each (I_i, M_i) is an attribute-set, m and n are integers such that $2 \leq k \leq min(m, n)$. This constraint forbids a user from $\{u_1, \cdots, u_n\}$ being a member of k or more attribute-sets in $\{(I_1, M_1), \cdots, (I_m, M_m)\}$.

While in a UCON_A system, there exists only one type of attribute, such as role, then the SMEA constraint can be expressed as $smea\,\langle R, U, k\rangle$, which is the general form of SMER constraint that it forbids a user from U being a member of k or more roles in R. In this way, we say SMEA covers SMER.

Definition 10. *A UCON_A state ε is safe with respect to an SMEA constraint $c = smea\langle\{(I_1, M_1), \cdots, (I_m, M_m)\}, U, k\rangle$, which we denote by $safe_c(\varepsilon)$, if and only if*

$$\forall u_i \in U(|(ATT(u_i) \cap \{(I_1, M_1), \cdots, (I_m, M_m)\})| < k)$$

$safe_c(\varepsilon)$ requires that no user is a member of k or more attribute-sets in $\{(I_1, M_1), \cdots, (I_m, M_m)\}$. A UCON_A state ε is safe with respect to a set C of SMEA constraints, which we denote by $safe_C(\varepsilon)$, if and only if ε is safe with respect to every constraint in C. As each SMEA constraint restricts the attribute-set memberships of a single user, it is efficient to check whether an UCON_A state satisfies a set of SMEA constraints, which provides a justification for using SMEA constraints to enforce SSoD policies.

Definition 11. *Given a UCON_A state ε, and a set C of SMEA constraints, determine whether $safe_C(\varepsilon)$ is true is called the safety checking problem for SMEA constraints (SC-SMEA).*

Theorem 3. SC-SMEA *is in P.*

Proof. One algorithm for verifying $safe_C(\varepsilon)$ is as follows. For each SMEA constraint in C and for each user in ε, one first computes the set of all the attribute-sets the user is a member of, then counts how many attribute-sets in this set also appear in the SMEA constraint, and finally compares this number with n. This algorithm has a time complexity of $O(N_u N_a N_c)$, where N_u is the number of users in ε, N_a is the number of attribute-sets in ε, and N_c is the number of constraints. □

SMEA constraints are expressed in terms of restrictions on attribute memberships, but SSoD policies are expressed in terms of restrictions on permissions. In order to generate SMEA constraints for enforcing SSoD policies, the first step is to translate restrictions on attribute-sets other than on permissions for SSoD policies. For each permission p_i in $\{p_1, \cdots, p_m\}$, there exists $(I, M)_{p_i}$ which is the threshold value of p_i. In this way, we can define the attribute-level SSoD requirement, and translate an SSoD policy to the attribute-level SSoD requirements.

Definition 12. *An attribute-level Static Separation-of-Duty (ASSoD) requirement is expressed as*

$$assod\langle\{(I_1, M_1), \cdots, (I_m, M_m)\}, \{u_1, \cdots, u_n\}, k\rangle$$

where each (I_i, M_i) is an attribute-set, m and n are integers such that $2 \leq k \leq min(m, n)$. It means that there should not exist a set of fewer than k users from $\{u_1, \cdots, u_n\}$ that together have memberships in all the m attribute-sets in the requirement.

Definition 13. *A UCON_A state ε is safe with respect to an ASSoD requirement $a = assod < \{(I_1, M_1), \cdots, (I_m, M_m)\}, \{u_1, \cdots, u_n\}, k >$, which we denote by $safe_a(\varepsilon)$, if and only if*

$$\forall\{u'_1, \cdots, u'_{k-1}\} \subseteq \{u_1, \cdots, u_n\}$$
$$\left(\bigcup_{i=1}^{k-1} auth\_p_\varepsilon(u'_i)\right) \nsubseteq \{(I_1, M_1), \cdots, (I_m, M_m)\}$$

A UCON_A state ε is safe with respect to a set A of ASSoD requirements if it is safe with respect to every requirement in A, and we write it as $safe_A(\varepsilon)$.

Definition 14. *Given a UCON_A state ε, and a set A of ASSoD requirements, determine whether $safe_A(\varepsilon)$ is true is called the safety checking problem for ASSoD constraints (SC-ASSoD).*

Theorem 4. SC-ASSoD *is coNP-complete.*

Proof. The proof is similar to the one in Theorem 2: let each attribute-set in ASSoD requirement map to a permission. Then the ASSoD requirement is mapped to an SSoD policy. □

6.2 Translating SSoD Policies to ASSoD Requirements

In this section, we show how to generate a set A of ASSoD requirements from an SSoD policy set E in a given

UCON_A state ε. By Theorem 1, if the SSoD policies in E are unenforceable, that means there must be at least one SSoD policy e in E of which cannot be satisfied by ε, such that there exists k-1 attribute set (I, M) together have all the permissions in $\{p_1, \cdots, p_m\}$. In this case, this SSoD policy cannot be translated to any ASSoD requirements. Let $ASSoD\langle A, \varepsilon\rangle$ denotes the set of ASSoD requirements derived from $\langle E, \varepsilon\rangle$, then the $ASSoD\langle A, \varepsilon\rangle$ is an empty set.

Otherwise, if the SSoD policies in E are enforceable in ε, then for each $e \in E$, where $e = ssod\langle\{p_1, \cdots, p_m\}, \{u_1, \cdots, u_n\}, k\rangle$, and for each permission p_i in $\{p_1, \cdots, p_m\}$, there exists an attribute-set (I_i, M_i) that we say (I_i, M_i) has the permission p_i . As shown in Example 1, $(\{engineer, student\}, \{50\})$ has p_1, $(\{programmer, student\}, \{150\})$ has p_2. Suppose that many attribute-sets may have only one permission, that p_3 is a permission, and that $allowed(u, p_3) \Rightarrow ATT(u) = \{engineer|programmer, student|teacher, 100\}$. It means that any user who wants to acquire the permission p_3 must have the the role of *engineer* or *programmer*, *Identity* of *student* or *teacher*, and the value of *Digital_money* is no less than 100. In this way, we divide it to four attribute-sets as follows:

- $\{\{engineer, student\}, \{100\}\}$,
- $\{\{engineer, teacher\}, \{100\}\}$,
- $\{\{programmer, student\}, \{100\}\}$,
- $\{\{programmer, teacher\}, \{100\}\}$.

As the above discussion shows, each permission may be related to one or more attribute-sets, assume that each permission in $\{p_1, \cdots, p_m\}$ relates to the number of attribute-sets is $\{k_1, \cdots, k_m\}$, then the total number of elements in $ASSoD\langle A, \varepsilon\rangle$ is $k_1 \times k_2 \times \cdots \times k_m$. Theorem 5 shows that for any enforceable SSoD policies, the $ASSoD\langle A, \varepsilon\rangle$ equals to it while both of them capture the same security requirement.

Theorem 5. *Given an SSoD policy set E, a UCON_A state ε, and the $ASSoD\langle A, \varepsilon\rangle$ derived from $\langle E, \varepsilon\rangle$, then $safe_A(\varepsilon) \Leftrightarrow safe_E(\varepsilon)$.*

Proof. Firstly, we show that if $safe_A(\varepsilon)$ is false, then $safe_E(\varepsilon)$ is also false. Assume that $safe_A(\varepsilon)$ is false, then there exist $a = assod\langle\{(I_1, M_1), \cdots, (I_m, M_m)\}, \{u_1, \cdots, u_n\}, k\rangle$ and k-1 users that together cover all attribute-sets in $\{(I_1, M_1), \cdots, (I_m, M_m)\}$. Given the way in which $ASSoD\langle A, \varepsilon\rangle$ is derived from $\langle E, \varepsilon\rangle$, there exists an SSoD policy in E such that the attribute-set in A together have all the permissions in it. Therefore, $safe_E(\varepsilon)$ is also false.

Secondly, we show that if $safe_E(\varepsilon)$ is false, then $safe_A(\varepsilon)$ is also false. If $safe_E(\varepsilon)$ is false, then there exists $e = ssod\langle\{p_1, \cdots, p_m\}, \{u_1, \cdots, u_n\}, k\rangle$ and k-1 users together cover all permissions in $\{p_1, \cdots, p_m\}$. For each permission p_i in the permission set, there exists an attribute-set (I_i, M_i) covers p_i, if it contains some sub attribute-set, then we divide it, then there exists $a = \langle\{(I_1, M_1), \cdots, (I_m, M_m)\}, \{u_1, \cdots, u_n\}, k\rangle$ derived from e. Given the way in which $ASSoD\langle A, \varepsilon\rangle$ is derived from $\langle E, \varepsilon\rangle$ then $a \in A$. Therefore,$safe_A(\varepsilon)$ is also false. □

6.3 Generating SMEA Constraints to Enforce ASSoD Requirements

Since Theorem 4 shows that it is intractable to direct statically enforce ASSoD requirements, while it is efficient to check whether an UCON_A state satisfies a SMEA constraint. Which motives us to generate SMEA constraints to indirect statically enforcing SSoD policies by using the ASSoD requirements as an intermediate step. We now show how to generate SMEA constraints from a set of ASSoD requirements.

Definition 15. *Let C be a set of SMEA constraints, and A be a set of ASSoD requirement. C enforce A if and only if $safe_C(\varepsilon) \Rightarrow safe_A(\varepsilon)$.*

Theorem 6. *For every ASSoD requirements, there exists a set of SMEA constraints that enforces it.*

Proof. One approach for constructing a set C of SMEA constraints to enforce a given ASSoD requirement $a = assod\langle\{(I_1, M_1), \cdots, (I_m, M_m)\}, \{u_1, \cdots, u_n\}, k\rangle$ as follows. We begin with $C = \emptyset$. Let I be the set of all immutable attributes in the state. For each nonempty subset S of I, such that all immutable attributes in S do not have a common ancestor, for every $s \in I$ such that $S \cup \{s\}$ does not have a common ancestor, add $smea\langle S \cup \{s\}, U, |S| + 1\rangle$ to C.

We now show that C enforces a, suppose, for the sake of contradiction that C does not enforce a, there exists k-1 users together have all the attributes in $\{(I_1, M_1), \cdots, (I_m, M_m)\}$ without violating any SMEA constraints in C. Then $\{(I_1, M_1), \cdots, (I_m, M_m)\}$ can be divided into k-1 sets which are the attribute memberships of the k-1 users. Then all attributes in each attribute set (I^*, M^*) must share a common ancestor, because if do not, then let $S \subset (I^*, M^*)$ be a largest subset of (I^*, M^*) that shares a common ancestor, let s be any attribute set in $(I^*, M^*)/S$, there must exists a SMEA constraint $smea\langle S \cup \{s\}, U, |S| + 1\rangle$ in C, and this constraint will be violated, which contradicting the assumption that all SMEA constraints in C will not be violated. □

The above theorem shows that we can generate SMEA constraints to enforce any ASSoD requirements. However, this may result in constraints that are more restrictive than necessary. Ideally, we want to generate SMEA constraints that can enforce the ASSoD requirement, and avoid generating constraints that are overly restrictive. For this, we give the definition of restrictive SMEA constraints.

Definition 16. *Let c_1 and c_2 be two sets of SMEA constraints, c_1 is at least as restrictive as c_2 (denoted by $c_1 \trianglerighteq c_2$), if and only if*

$$\forall \varepsilon (safe_{\{c_1\}}(\varepsilon) \Rightarrow safe_{\{c_2\}}(\varepsilon))$$

when $c_1 \trianglerighteq c_2 \wedge c_2 \not\trianglerighteq c_1$, we say that c_1 is more restrictive than c_2 (denoted by $c_1 \triangleright c_2$), and when both $c_1 \trianglerighteq c_2 \wedge c_2 \trianglerighteq c_1$, we say c_1 is equivalent to c_2(denoted by $c_1 \triangleq c_2$), otherwise, we say are incomparable(denoted by $c_1 \not\triangleq c_2$).

For every $smea < \{(I_1, M_1), \cdots, (I_m, M_m)\}, \{u_1, \cdots, u_n\}, k > (k < m)$, we then show that it can be equivalently represented using a set of canonical ($k = m$) SMEA constraints.

Lemma 4. *For every SMEA constraint c ($k < m$), there exists a set $C^{'}$ of canonical SMEA constraints of cardinality k such that $C^{'} \triangleq c$.*

Proof. Given a SMEA constraint $c = smea < \{(I_1, M_1), \cdots, (I_m, M_m)\}, \{u_1, \cdots, u_n\}, k > (k < m)$, Let $C^{'}$ be

$$\{smea\,\langle (I', M'), \{u_1, \cdots, u_n\}, k\rangle \} | (I', M') \subset \{(I_1, M_1), \cdots, (I_m, M_m)\} \wedge |(I', M')| = k$$

$C^{'}$ is the set of all $smea\,\langle (I', M'), \{u_1, \cdots, u_n\}, k\rangle$ constraints, such that (I', M') is a size-k subset of $\{(I_1, M_1), \cdots, (I_m, M_m)\}$. 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, $C^{'} \triangleq c$. □

Given a set of ASSoD requirements, there are multiple SMEA constraint sets can enforce these ASSoD requirements. We prefer to use the less restrictive constraint set. In the following, we show how to compare two sets of SMEA constraints. As shown in Lemma 4, in this paper, we treat a SMEA constraint ($k < m$) as a set of canonical SMEA constraints ($k = m$), and compare two sets of canonical SMEA constraints. We use $junior(I, M)$ to denote the set of all attribute sets that are junior to some attribute sets in (I, M).

Lemma 5. *For any state ε and any canonical SMEA constraints $c_i = smea\langle (I_i, M_i), u, k_i\rangle$ and $c_j = smea\langle (I_j, M_j), u, k_j\rangle$, $c_i \triangleright c_j$ if and only if $junior(I_i, M_i) \subset junior(I_j, M_j)$.*

Proof. For the "only if" part, Suppose, for the sake of contradiction, that $c_i \triangleright c_j$ and $junior(I_i, M_i) \not\subset junior(I_j, M_j)$. Consider a singer user in a state ε, and the user is assigned to all attribute sets in (I_j, M_j), then $safe_{c_j}(\varepsilon)$ is false, but $safe_{c_i}(\varepsilon)$ is true since the single user is not authorized for all attribute sets in (I_i, M_i). However, this contradicts the assumption that $c_i \triangleright c_j$.

For the "if" part, if $safe_{c_j}(\varepsilon)$ is false, then there exists a user who is authorized for all attribute sets in (I_j, M_j), and this user is also authorized for all attribute sets in (I_i, M_i). Therefore, $\forall \varepsilon (\neg safe_{c_j}(\varepsilon) \Rightarrow \neg safe_{c_i}(\varepsilon))$. □

It is efficient to decide if $c_i \triangleright c_j$, thus we can remove the redundant SMEA constraints, until we cannot remove any constraint, this should give a least restrictive constraint set. The following definition makes this more precise.

Definition 17. *Let A be a set of ASSoD requirements and C be a set of SMEA constraint, C is the least restrictive SMEA of A if and only if $safe_{\{C\}}(\varepsilon) \Rightarrow safe_{\{A\}}(\varepsilon)$, and there does not exist a different set $C^{'}$ of SMEA constraints such that $safe_{\{C^{'}\}}(\varepsilon) \Rightarrow safe_{\{A\}}(\varepsilon)$ and $C^{'} \subset C$.*

It can be easily shown that there exist two special cases where there exist least restrictive SMEA constraints to enforce these ASSoD requirements: (1) $smea < \{(I_1, M_1), \cdots, (I_m, M_m)\}, \{u_1, \cdots, u_n\}, 2 >$ is the least restrictive SMEA enforces $assod < \{(I_1, M_1), \cdots, (I_m, M_m)\}, \{u_1, \cdots, u_n\}, 2 >$. (2) $smea < \{(I_1, M_1), \cdots, (I_m, M_m)\}, \{u_1, \cdots, u_n\}, 2 >$ is the least restrictive SMEA enforces the $assod < \{(I_1, M_1), \cdots, (I_m, M_m)\}, \{u_1, \cdots, u_n\}, 2 >$. The proof are essentially the same as that for Lemma 2 and Lemma 3 respectively.

Given an ASSoD requirement, where $2 < k < m$, there may exist many sets of least restrictive SMEA constraints that enforce it. We give an algorithm to generate a least restrictive SMEA constraint set to enforce ASSoD requirement in Algorithm 2. The algorithm first constructs a set C of SMEA constraints to enforce a given ASSoD requirement (from step 1 to 10). Secondly, the algorithm tries to remove the redundant constraints in C, until we cannot remove any constraints (from step 11 to 15). Finally, the algorithm generates all such least restrictive constraints. The algorithm has a time complexity of $O(m^2 \cdot k \cdot |AH|)$ for step 1 to 10, while m and k are the two integers in in the ASSoD requirement, and $|AH|$ is the attribute hierarchy in $\{(I_1, M_1), \cdots, (I_m, M_m)\}$, the time complexity is none of n. The algorithm has a time complexity of $O(|C|^2 \cdot |AH|)$ for step 11 to 15, while C is the SMEA constraint set.

We now give an example to show how to generate SMEA constraints that are least restrictive in implementing an ASSoD requirement in a UCON_A state.

Example 2. *Continuing from Example 1, assume that an SSoD policy is $e = ssod\langle \{p_1, p_2, p_3, p_4\}, U, 3\rangle$, and each user requesting the permission p_3 or p_4 must comply with the following preA predicates:*

- *allowed(u,p_3)⇒ preA(ATT(u),p_3), where ATT(u)= {{programmer,teacher},{100}}*

Algorithm 2. Generating a least restrictive SMEA constraint set for an ASSoD requirement.

Input: $assod\langle\{(I_1, M_1), \cdots, (I_m, M_m)\}, \{u_1, \cdots, u_n\}, k\rangle$,
Output: C //a least restrictive SMEA constraints for an ASSoD requirement.
1: initialize $C = \emptyset$;
2: **foreach** $\emptyset \subset S \subseteq \{(I_1, M_1), \cdots, (I_m, M_m)\}$
3: **if** attributes in S do not share an ancestor
4: **foreach** $s \subseteq \{(I_1, M_1), \cdots, (I_m, M_m)\}$
5: **if** attributes in $S \cup \{s\}$ do not share an ancestor
6: $C = C \cup \{c\}$
7: **end if**
8: **end foreach**
9: **end if**
10: **end foreach**
11: **foreach** $c_i \in C$, $c_j \in C$ and $c_i \neq c_j$
12: **if** $c_i \rhd c_j$
13: $C = C/c_j$
14: **end if**
15: **end foreach**
16: **return** C

- *allowed(u,p4)⇒ preA(ATT(u),p4), where ATT(u)= {{supervisor,teacher},{200}}*

For the sake of simplicity we refer the total four preA predicates as follows:

- $p_1 \rightarrow (I_1, M_1) = \{\{engineer, student\}, \{50\}\}$
- $p_2 \rightarrow (I_2, M_2) = \{\{programmer, student\}, \{150\}\}$
- $p_3 \rightarrow (I_3, M_3) = \{\{programmer, teacher\}, \{100\}\}$
- $p_4 \rightarrow (I_4, M_4) = \{\{supervisor, teacher\}, \{200\}\}$

Firstly, we translate the SSoD policy to ASSoD requirements:

- $a = assod\langle\{(I_1, M_1), (I_2, M_2), (I_3, M_3), (I_4, M_4)\}, U, 3\rangle$

Secondly, we generate a SMEA constraints for A by Algorithm 1 (from step 1 to step 10). (I_1, M_1) and (I_2, M_2) share a common ancestor $\{\{supervisor, student\}, \{150\}\}$, and (I_3, M_3) and (I_4, M_4) share a common ancestor $\{\{supervisor, teacher\}, \{200\}\}$. The generating SMEA constraints as follows:

- $c_1 = smea\langle\{(I_1, M_1), (I_3, M_3)\}, U, 2\rangle$
- $c_2 = smea\langle\{(I_1, M_1), (I_4, M_4)\}, U, 2\rangle$
- $c_3 = smea\langle\{(I_2, M_2), (I_3, M_3)\}, U, 2\rangle$
- $c_4 = smea\langle\{(I_2, M_2), (I_4, M_4)\}, U, 2\rangle$

Thirdly, we remove the redundant constraints in C by Algorithm 2 (from step 11 to step 15). Since $c_2 \rhd c_1$ and $c_4 \rhd c_3$, then the constraints c_1 and c_3 can be removed. Therefore, the set of following constraints is a least restrictive SMEA for a.

- $c_2 = smea\langle\{(I_1, M_1), (I_4, M_4)\}, U, 2\rangle$
- $c_4 = smea\langle\{(I_2, M_2), (I_4, M_4)\}, U, 2\rangle$

7. Related Work

The concept of SoD dated back to 1975 by Saltzer and Schroeder [14]; they took it as one of the design principles for protecting information, under the name "separation-of-privilege". The research community has taken an active interest in incorporating separation of duty controls into computer systems since the late 1980s, Clark and Wilson [1] applied SoD principle to data objects to ensure integrity and to control frauds along with well-formed transactions as two major mechanisms for controlling fraud and error. Later on, SoD has been studied by various researchers as a security principle to avoid frauds. There are two important issues relating to SoD policies: specification and enforcement.

As for specification, it should be noted that most existing approaches to SoD only consider constraint sets with precisely two elements, the exception being the RCL 2000 specification language [15]. And the distinction between the SSoD policy objectives and the SMER constraints, as a mechanism to enforce them, is sometimes not clearly made; Ferraiolo et al. defined SSoD 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." [16]. The specification scheme of the SSoD policy we propose has its basis in set-based approach to conflict of interest, and it is considerably simpler syntactically than other schemes because the SSoD policy is expressed in terms of restrictions on permissions other than attributes, such as roles. And we make no attempt to define the conditions that must be met for the constraints to be satisfied. In this way, our specification of SSoD is a generalization in the context of UCON_A.

In terms of enforcement, there are many approaches to enforce an SoD policy. One approach is dynamic enforcement. For example, in the literature [17], DSoD constraint prevents a user from simultaneously activating mutually exclusive roles in a session. It can be enforced by maintaining a history which included information on who performed each step. Sandhu [18], [19] adopted this idea and presented transaction control expressions, a history based mechanism for dynamically enforcing SoD policies. Simon and Zurko [4] combined the Object SoD and Operational SoD and introduced a notion of history based SoD. The alternative is to keep a history record of all access requests and to enter either p_1 or p_2 at some point in the future. Crampton [20] employed blacklist to enforce historical constraints, it do not need to keep a historical record. However, since these approaches for SoD only consider constraint sets with a few elements, they will have unacceptable overheads to support large range of constraints.

Another approach to enforce SoD policies is static enforcement. The first paper on SoD policies in RBAC is proposed by Ferraiolo and Kuhn [21], who used the terms static and dynamic SoD to refer to static and dynamic enforcement of SoD, to our knowledge. It is widely believed that one of RBAC's main strength is that it enables the use of constraints to support SoD

policies. In the static enforcement of SoD policies, there exists a wealth of literature on constraints to enforce SSoD policies in RBAC, for example, mutual exclusion of permissions, roles or users. Kuhn [22] discussed mutual exclusion of roles for SoD policies 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. Li et al. [11] used static mutually exclusive role (SMER) constraints to enforce SSoD policies in RBAC.

However, UCON_A includes traditional access control models, such as discretionary access control (DAC)[23], mandatory access control (MAC) [24], and role-based access control (RBAC) [25], it was proposed as a general and comprehensive model to extend the underlying mechanism of traditional access control models. The enforcement of SoD policies in UCON is of course more difficult than in traditional access control models, because authorization decisions in UCON are not only checked and made before the access, but may be repeatedly checked during the access and may be revoked if some policies are not satisfied, according to the changes of the subject or object attributes, or environmental conditions. Motivated by the SMER constraints, we introduce the concept of SMEA constraints, where SMEA is an extension to SMER, and it is also more complex than SMER, since the attribute set consists of the set of immutable attributes and the set of mutable attributes. Which enables to distinguishing properties of decision continuity and attribute mutability for UCON. It is efficient to check whether a UCON_A state satisfies a set of SMEA constraints, which provides a justification for using SMEA constraints to enforce SSoD policies. Therefore, we generate the least restrictive SMEA constraints for enforcing SSoD policies in UCON_A, by using the attribute-level SSoD requirement as an intermediate step. The results are fundamental to understanding the enforcement of SSoD policies in UCON.

8. Conclusion and Future Work

We have studied the fundamental problem of static enforcement of SSoD policies in UCON_A. We first give a set-based formal specification of SSoD in the context of UCON_A systems. We also show that static enforcement is a simple and straightforward enforcement mechanism for SSoD policies compare to dynamic enforcement. For the static enforcement aspect, we show that direct statically enforcing SSoD policies in UCON_A system is intractable (coNP-complete), while enforcing SMEA constraints is efficient, which provides a justification for using SMER constraints to enforce SSoD policies. Therefore, we translate SSoD policies to ASSoD requirements that use ASSoD as an intermediate step, and generate the least restrictive SMEA constraints from a set of ASSoD requirements. We also study the problem how to verify whether a given SSoD configuration is enforceable. The results are fundamental to understanding the effectiveness of using constraints to enforce SSoD policies in UCON.

This paper only focuses on the SSoD policy with pre-authorization policies in UCON. For condition core models of UCON, and on-authorization policies in UCON, it is a difficult problem because the system state changes nondeterministically in UCON_onA. As monitoring is actively involved in usage decisions while a requested right is exercised, the first approach in this paper will be most promising approach for enforcing SSoD in UCON_onA system. We leave the research of the SSoD policies in UCON_onA models, and the DSoD policies for future work.

9. Acknowledgements

This work is supported by National Natural Science Foundation of China under Grant 60873225 and 61170108, Zhejiang Province Education Foundation under Grant No.Y201120897.

References

[1] D. D. Clark, D.R. Wilson.: A comparison of commercial and military computer security policies. 8th IEEE Symposium on Security and Privacy, Los Alamitos, 1987, pp.184-195.

[2] D. D. Clark, D.R. Wilson.: Evolution of a model for computer Integrity. Technical Report, Invitational Workshop on Data Integrity, Section A2,1989, pp.1-3.

[3] D. Brewer, M. Nash.: The Chinese wall security policy. 10th IEEE Symposium on Security and Privacy, California, 1989, pp.206-214.

[4] R. T. Simon, M. E. Zurko.: Separation of Duty in Role-Based Environments. 10th Computer Security Foundations Workshop, June 10-12, 1997, pp.183-194.

[5] V.D. Gligor, S.I. Gavrila, D. Ferraiolo: On the Formal Definition of Separation- of-Duty Policies and their Composition. 19th IEEE Symposium on Security and Privacy, 1998, pp.172-183.

[6] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli.: Role-Based Access Control, Artech House, 2003, pp.47-63.

[7] J. Park., R. Sandhu.: The UCONABC Usage Control Model. ACM Transactions on Information and System Security. Vo. 7, No. 1, 2004, pp.128-174.

[8] X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park.: Formal model and policy specification of usage control. ACM Trans. Inf. Syst. Security, vol. 8, no. 4, pp.351-387, 2005.

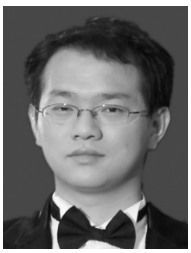
[9] N. Li, J. C. Mitchell, and W. H. Winsborough.: Beyond Proof-of-Compliance: Security Analysis in Trust Management. Journal of the ACM. Volume 52, Number. 3, 2005, pp.474-514.

[10] ANSI. American National Standard for Information Technology-Role Based Access Control. ANSI INCITS 359-2004, 2004.

[11] N. Li., M. Tripunitara., and Z. Bizri.: On Mutually Exclusive Roles and Separation-of-Duty. ACM Transactions on Information and System Security. Vol. 10, No. 2, 2007, pp.1-35.

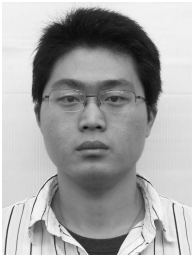
[12] N. Li, M. V. Tripunitara, and Q. Wang.: Resiliency Policies in Access Control. 13th ACM Conference on Computer

and Communication Security (CCS), Alexandria, Virginia, USA, 2006, pp.113-123
[13] C. H.Papadimitriou. Computational Complexity. Addison Wesley Longman, 1994.
[14] J. H. Saltzer, M. D. Schroeder.: The Protection of Information in Computer Systems. Proceed Communications of the ACM. Vol. 63, No. 9, 1975, pp.1278-1308.
[15] G. J. Ahn, R. Sandhu.: Role-based authorization constraints specification. ACM Transactions on Information and System Security. Volume 3, Number 4, 2000, pp.207-226.
[16] D. F. Ferralolo, J. A. Cuigini, and D. R. Kuhr. 1995. Role-based access control (RBAC): Features and motivations. Annual Computer Security Applications Conference, 1995.
[17] S. N. Foley. The specification and implementation of 'commercial' security requirements including dynamic segregation of duties. 4th ACM Conference on Computer and Communications Security. 1997, pp.125-134.
[18] R. Sandhu.: Transaction Control Expressions for Separation of Duties. 4th Annual Computer Security Applications Conference, Orlando, Florida, 1988, pp. 282-286.
[19] R. Sandhu.: Separation of Duties in Computerized Information Systems. the IFIP WG11.3 Workshop on Database Security, Halifax, 1990, pp. 18-21.
[20] J. Crampton.: Specifying and enforcing constraints in role-based access control. 8th ACM Symposium on Access Control Models and Technologies, Como, Italy, 2003, pp. 43-50.
[21] D. F. Ferraiolo, and D. R. Kuhn.: Role-based access control. 15th National Information Systems Security Conference,1992.
[22] 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, 1997, pp. 23-30.
[23] M. H. Harrison, W. L. Ruzzo, and J. D. Ullman.: Protection in operating systems. Commun. ACM, vol. 19, no. 8, 1976, pp. 461-471.
[24] D. E. Denning.: A lattice model of secure information flow. Commun. ACM, vol. 19, no. 5, 1976, pp. 236-243.
[25] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman.: Role-Based Access Control Models. Comput., Vol. 29, No. 2, 1996, pp. 38-47.



Jianfeng Lu received the B.S. degree in School of Computer Science and Technology at Wuhan University of Science and Technology in 2005, and the PhD degree in School of Computer Science and Technology at Huazhong University of Science and Technology in 2010. He is a lecturer in the School of Mathematics-Physical and Information Engineering at Zhejiang Normal University. His research interests include distributed system security and access control.

Ruixuan Li received the B.S., M.S., and Ph.D. degrees from School of Computer Science and Technology at Huazhong University of Science and Technology in 1997, 2000, and 2004, respectively. He is now a full Professor of School of Computer Science and Technology at Huazhong University of Science and Technology. His research interests include distributed system security, information retrieval, peer-to-peer computing, and social network.



Jinwei Hu received his B.S. degree in School of Information Engineering from Nanchang University in 2004, and the PhD degree in College of Computer Science and Technology at Huazhong University of Science and Technology in 2010. He is now a post-doc in Department of Computer Science, College of Engineering, Qatar University. His research interests include policy analysis, distributed system security, and access control.

Dewu Xu received his M.S. degree in School of Information Science and Technology from East China Normal University in 2005. Now he is a lecturer in the School of Mathematics-Physical and Information Engineering at Zhejiang Normal University. His research interests include distributed system security.