PAPER Special Section on Trust, Security and Privacy for Pervasive Applications

Consistency Checking of Safety and Availability in Access Control

Ruixuan LI[†a)], ***Member***, **Jianfeng LU**[†], **Zhengding LU**[†], ***and*** **Xiaopu MA**[†], ***Nonmembers***

SUMMARY The safety and availability policies are very important in an access control system for ensuring security and success when performing a certain task. However, conflicts may arise between safety and availability policies due to their opposite focuses. In this paper, we address the problem of consistency checking for safety and availability policies, especially for the co-existence of static separation-of-duty (SSoD) policies with availability policies, which determines whether there exists an access control state that satisfies all of these policies. We present criteria for determining consistency with a number of special cases, and show that the general case and partial subcases of the problem are intractable (NP-hard) and in the Polynomial Hierarchy NP^{NP}. We design an algorithm to efficiently solve the nontrivial size instances for the intractable cases of the problem. The running example shows the validity of the proposed algorithm. The investigation will help the security officer to specify reasonable access control policies when both safety and availability policies coexist.

key words: *access control, availability, consistency checking, safety, separation-of-duty*

1. Introduction

An access control policy focuses on safety properties ensuring that users who should not have an access do not get the access. Safety analysis has been studied extensively in the context of access control systems [1], [2]. One example of an access control policy focuses on safety properties is a separation-of-duty (SoD) policy, which is considered as a fundamental principle of information security that has been widely used in business, industry, and government applications [3]. A static SoD (SSoD) policy states that the cooperation among at least a certain number of users is required in order to have all permissions necessary to complete a sensitive task. There exists a wealth of literature on SoD policies, and almost all existing work focuses on safety properties. For example, Crampton [4] developed a simple set-based specification scheme for SoD in role-based access control systems, and suggested an enforcement model for a restricted subset of this scheme. Such focus on safety properties can be viewed as a tool for restricting access.

In the meantime, to ensure the tasks to be performed smoothly, it is also important to analyze the maximum number of users needed to complete a task [5]. An equally important aspect of access control is the availability property that enabling access in the context of access control systems. The availability policy requires that the cooperation among at most a certain number of users is necessary to perform a task. Without the availability requirement, an access control state can trivially satisfy a safety requirement of an SSoD policy if the state does not contain any user set that covers all the permissions needed to accomplish the sensitive task. In particular, an empty access control state satisfies any safety requirements of SSoD policies. Similarly, without the safety requirement, the availability requirement can be satisfied by giving all permissions to all users, which allows each single user to accomplish any task.

SSoD policies focusing on safety properties can be viewed as a tool for restricting access, while availability policies focusing on availability properties can be viewed as a tool for enabling access. There may exist conflicts between SSoD policies and availability policies. In the example of ordering and paying for goods given by Clark and Wilson [3], there are four steps as follows: (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 on the order; (3) verifying that the goods have been received and the features of the goods match the details on the invoice; and (4) authorizing payment to the supplier against the invoice. Let's assume that there are four users $\{Alice, Bob, Carl, Doris\}$ who prepare to accomplish the task.

The security officer may define many policies that require safety and availability properties in this example and these policies may be inconsistent. For example, he may define two policies, one requiring that no user can perform all the four steps while the other allowing the existence of a user who can perform all the four steps. Clearly, the two policies are inconsistent as they cannot be satisfied simultaneously. However, it is not easy to check whether a set of policies with the safety and availability requirements is consistent. In practice, the security officer may define a number of policies with the safety and availability requirements. For example, with respect to safety requirements, he may require that (a) the cooperation of at least three users is needed to perform all four steps, (b) no single user can perform both step (1) and (2), (c) no single user from $\{Bob, Carl, Doris\}$ can perform both step (3) and (4), and (d) no less than three users can perform all of the step (1), (2) and (3). On the other side, as to availability requirements, he may require that (e) at most three users together perform all four steps, (f) either Alice or Bob needs to perform both steps (1) and (4), and (g) at most two users from $\{Alice, Bob, Carl\}$ need

Manuscript received July 3, 2009.
Manuscript revised October 7, 2009.

[†]The authors are with College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, P. R. China.
a) E-mail: rxli@hust.edu.cn

DOI: 10.1587/transinf.E93.D.491

to perform all of the steps (1), (2) and (3). Checking whether these requirements are consistent is thus not straightforward. (We will show that these requirements of safety and availability are consistent in Sect. 4.3.)

In many cases, it is desirable for an access control system to satisfy both safety and availability requirements. However, these policies may conflict with each other due to their opposite focuses. Therefore, it is very important to study the problem of consistency checking for the co-existence of safety policies with availability policies, which determines whether there exists an access control state satisfying all of these policies. The investigation will help the security officer to specify reasonable access control policies when both safety and availability policies coexist. Our contributions in this paper are as follows:

- We formally define the SSoD policies and availability policies, and the consistency checking problem for the co-existence of SSoD policies with availability policies, which express requirements about safety and availability in the context of access control systems.
- We study the computational complexities of the consistency checking problem, and present criteria for determining consistency for a number of special cases, and show that the general case of the problem and several subcases are intractable (NP-hard) and in the Polynomial Hierarchy (NP^{NP}).
- We present an algorithm for consistency checking problem. The algorithm uses a pruning technique that reduces the number of combinations that need to be considered. Our algorithm also takes advantage of existing SAT solvers. The experimental results show that our algorithm can efficiently solve instances of nontrivial sizes. And the running example shows the validity of our algorithm.

The rest of this paper is organized as follows. In Sect. 2, we define SSoD policies and availability policies, and consistency checking problem. In Sect. 3, we study computational complexities of the consistency checking problem. Section 4 proposes an algorithm to verify that the problem can be solved in reasonable amount of time for general cases. And the evaluation and illustration of the algorithm will be given in Sect. 5. We discuss related work in Sect. 6. Finally, we summarize this paper and present some ongoing and future work in Sect. 7.

2. Safety and Availability Checking Problems

In this section, we give formal definitions for SSoD policies, availability policies, and the consistency checking problem for their co-existence.

2.1 Static Separation-of-Duty (SSoD) Policies

An SSoD policy typically constrains the assignment of permissions to users, which precludes any group of users from possessing too many permissions. The concrete formulation of SSoD policy in the context of access control must base on the following requirements [6]:

- An SSoD policy must be a high-level requirement.
- An SSoD policy must be described in terms of restrictions on permissions.
- An SSoD policy must capture restrictions on user set involved in the task.

Definition 1. *An SSoD policy ensures that at least k users from a user set are required to perform a task that requires all these permissions. It is formally defined as*

- *P and U denote the set of permissions and the set of users, respectively.*
- $UP \subseteq U \times P$, *is a user-permission assignment relation.*
- $auth\_P_{UP}[u] = \{p \in P | (u, p) \in UP\}$.
- $\forall (P, U, k) \in SSoD, \forall U' \subseteq U : |U'| < k \Rightarrow \bigcup_{u \in U'} auth\_p_{UP}(u) \not\supseteq P$.

Where $P = \{p_1, \cdots, p_m\}$, $U = \{u_1, \cdots, u_n\}$, *each* p_i *in P is a permission,* u_j *in U is a user, m, n, and k are integers, such that* $2 \leq k \leq min(m, n)$, *min returns the smaller value of the two. We write an SSoD policy as* $ssod\langle P, U, k\rangle$.

SSoD policies can be defined in any access control system in which there are users and permissions. We assume that an access control state ϵ is given by a binary relation $UP \subseteq U \times P$, where U denotes the set of users, and P denotes the set of all permissions. UP determines the permission to user assignment relation. We say that an access control state ϵ is safe with respect to an SSoD policy $e = ssod\langle\{p_1, \cdots, p_m\}, \{u_1, \cdots, u_n\}, k\rangle$, if in state ϵ no $k-1$ users from $\{u_1, \cdots, u_n\}$ together have all the permissions in $\{p_1, \cdots, p_m\}$, and we write it as $safe_e(\epsilon)$. 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. An access control state ϵ is safe with respect to a set E of SSoD policies, which we denote by $safe_E(\epsilon)$, if and only if, ϵ is safe with respect to every policy in the set E.

Definition 2. *Given an access control state* ϵ *and a set E of SSoD policies, determine whether* $safe_e(\epsilon)$ *is true is called the safety checking problem (SCP).*

Theorem 1. *SCP is coNP-complete.*

Proof. Consider the complement of SCP, i.e., given an access control state ϵ and a set E of SSoD policies, determine if $safe_E(\epsilon)$ is false, which is denoted by $\overline{\text{SCP}}$.

We first show that $\overline{\text{SCP}}$ 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 $safe_e(\epsilon)$, one only

needs to compute the set of permissions of every *size*-(k-1) user 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, $\overline{\text{SCP}}$ is in NP.

We now show that $\overline{\text{SCP}}$ is NP-hard by reducing the NP-complete set covering problem [7] to $\overline{\text{SCP}}$. 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 an SSoD policy $e = ssod\langle S, \{u_1, \cdots, u_n\}, B+1\rangle$, and construct an access control state as follows. For each different subset $S_i (1 \le i \le l)$ in F, create a user $u_i \in \{u_1, \cdots, u_n\}$, to which all permissions in S_i are assigned. The resulting SSoD configuration is not enforceable if and only if B sets in F cover S. □

2.2 Availability Policies

In order to ensure the tasks to be performed smoothly, it is also important to analyze the maximum number of users needed to complete a task. Take the example that we discussed in Sect. 1, if all of the users in {*Alice*, *Bob*, *Carl*, *Doris*} cannot work together to perform all the four steps, then the task cannot be completed in the current system configuration, although the safety requirement that the cooperation of at least three users in {*Alice*, *Bob*, *Carl*, *Doris*} is needed to perform all the four steps can be satisfied. Obviously, it is probably not what the designers of the safety policies desire. In this way, we introduce the notion of availability policies which express requirements about enabling access rather than restricting access.

Definition 3. *An availability policy (AP) ensures that at most t users from U are required to complete a task that requires all these permissions in P. It is formally defined as*

- *P and U denote the set of permissions and the set of users, respectively.*
- $UP \subseteq U \times P$, *is a user-permission assignment relation.*
- $auth\_P_{UP}[u] = \{p \in P | (u, p) \in UP\}$.
- $\forall (P, U, k) \in AP, \exists U' \subseteq U : |U'| \le t \Rightarrow \bigcup_{u \in U'} auth\_p_{UP}(u) \supseteq P$.

Where $P = \{p_1, \cdots, p_m\}$, $U = \{u_1, \cdots, u_n\}$, *each* p_i *in P is a permission,* u_j *in U is a user, m, n, and k are integers, such that* $2 \le k \le min(m, n)$, *min returns the smaller value of the two. We write an availability policy as* $ap\langle P, U, t\rangle$.

Availability policies can also be defined in any access control system in which there are users and permissions. The permissions in an availability policy are the permissions needed to carry out a sensitive task and the policy guarantees that at most t users from $\{u_1, \cdots, u_n\}$ are required to perform a task that requires all these permissions in $\{p_1, \cdots, p_m\}$. It is necessary to make sure that the task can be completed, especially when it is a critical task. We say that an access control state ϵ satisfies an availability policy $f = ap\langle\{p_1, \cdots, p_m\}, \{u_1, \cdots, u_n\}, t\rangle$, if in state ϵ there exists a subset of $\{u_1, \cdots, u_n\}$ of size no more than t that together have all the permissions in $\{p_1, \cdots, p_m\}$, and we write it as $sat_f(\epsilon)$. Observe that if no t users together have all the permissions in a policy, then no set of fewer than t users together can also have all the permissions. An access control state ϵ satisfies a set F of availability policies, which we denote by $sat_F(\epsilon)$, if and only if ϵ is safe with respect to ever policy in the F.

Definition 4. *Given an access control state* ϵ *and a set F of availability policies, determine whether* $sat_F(\epsilon)$ *is true is called the availability checking problem (ACP).*

Theorem 2. *ACP is NP-complete.*

Proof. We first show that ACP is in NP. $sat_F(\epsilon)$ is true if and only if every availability policies in F can be satisfied in an access control state ϵ. For each $f_i = ap\langle P_i, U_i, t_i\rangle$ in F, which mandates that the users in U_i together cover all permissions in P_i and each set has at most t_i users. If the set of t_i users is given, the verification can be done in polynomial time: compute the union of the t_i users' permissions and check whether it is a superset of P_i.

We now show that ACP is NP-hard by reducing the NP-complete set covering problem to ACP. The proof is similar to the proof that $\overline{\text{SCP}}$ is NP-hard in Theorem 1: For each $f_i = ap\langle P_i, U_i, t_i\rangle$ in F, each permission in P_i is mapped to an element in S, each user u_i in U_i is mapped to a subset S_i, and let t_i be B. □

2.3 Consistency Checking Problem for SSoD and Availability Policies

SSoD policies and availability policies may conflict with each other due to their opposite focuses. In this section, we address the problem of consistency checking for SSoD and availability policies. Obviously, availability policies are a natural complement to SSoD policies in access control. Neither kind of policies by itself is sufficient to capture both safety and availability requirements. When SSoD and availability policies coexist, both safety and availability requirements may not be able to be satisfied simultaneously. Consequently, we formally define our notion of the consistency between SSoD and availability policies. (We will study the computational complexity of the consistency checking problem in Sect. 3.)

Definition 5. *Given a set E of SSoD policies and a set F of availability policies, determine whether there exists an access control state* ϵ *that* $safe_E(\epsilon) \wedge sat_F(\epsilon)$ *is true is called the consistency checking problem (CCP).*

Let $Q = E \cup F$, if Q is inconsistent, it implies that there does not exist an access control state ϵ such that $safe_E(\epsilon) \wedge sat_F(\epsilon)$ is true. Thus, for any state ϵ, there exists at least an SSoD policy e in E and an availability policy f in F such that $safe_e(\epsilon) \wedge sat_f(\epsilon)$ is false. Let $e = ssod\langle P, U, k\rangle$, $f = ap\langle P', U', t\rangle$, the degree to which permissions are shared by P and P', and users are shared by U and U', four possibilities may be considered.

- Disjoint/Disjoint (D/D): Permission sets for SSoD and availability policies are disjoint, user sets for SSoD and availability policies are also disjoint.

$$P \cap P' = \emptyset \wedge U \cap U' = \emptyset$$

- Disjoint/Shared (D/S): Permission sets for SSoD and availability policies are disjoint, user sets for SSoD and availability policies are shared.

$$P \cap P' = \emptyset \wedge U \cap U' \neq \emptyset$$

- Shared/Disjoint (S/D): Permission sets for SSoD and availability policies are shared, user sets for SSoD and availability policies are disjoint.

$$P \cap P' \neq \emptyset \wedge U \cap U' = \emptyset$$

- Shared/Shared (S/S): Permission sets for SSoD and availability policies are shared, user sets for SSoD and availability policies are also shared.

$$P \cap P' \neq \emptyset \wedge U \cap U' \neq \emptyset$$

In fact, not all these four possibilities must be considered, only the fourth one. Theorem 3 asserts that if there is an SSoD policy $e_i = ssod\langle P_i, U_i, k_i\rangle$, where there exists a permission $p \in P_i \wedge p \notin R$ (R denotes the union of permissions other than e_i in all policies in Q), then e_i does not affect the consistency of $Q-\{e_i\}$. Similarly, if there exists an availability policy $f_j = ap\left\langle P_j^{'}, U_j^{'}, t_j\right\rangle$, where there exists a user $u \in U_j^{'} \wedge u \notin S$ (S denotes the union of users in all policies other than f_j in Q), then f_j does not affect the consistency of Q-$\{f_i\}$.

Theorem 3. *Let $Q = \{e_1, \cdots, e_m, f_1, \cdots, f_n\}$, where $e_i = ssod\langle P_i, U_i, k_i\rangle$ $(1 \le i \le m)$, $f_j = ap\left\langle P_j^{'}, U_j^{'}, t_j\right\rangle$ $(1 \le j \le n)$. If $\exists e_a \in Q(P_a - R \neq \emptyset)$, where $R = \bigcup_{i=1,i\neq a}^{m} P_i \cup \bigcup_{j=1}^{n} P_j^{'}$, then let $Q^{'} = Q' - \{e_a\}$; If $\exists f_b \in Q(U_b^{'} - S \neq \emptyset)$, where $S = \bigcup_{i=1}^{m} U_i \cup \bigcup_{j=1,j\neq b}^{n} P_j^{'}$, then let $Q^{'} = Q' - \{f_b\}$; Q is consistent if and only if $Q^{'}$ is consistent.*

Proof. It is clear that if Q is consistent then $Q^{'}$ is consistent as $Q^{'} \subseteq Q$. We now show that if $Q^{'}$ is consistent then Q is consistent. $Q^{'}$ is consistent implies that there exists an access control state ϵ satisfies all policies in $Q^{'}$. We now construct a new state $\epsilon^{'}$ that satisfies both $Q^{'}$ and Q as follows: for each $e_i \in Q/Q^{'}$, where $e_i = ssod\langle P_i, U_i, k_i\rangle$, add all users in U_i to ϵ, but do not assign any permissions in $P_i \cap R^{'}$ ($R^{'} = \bigcup_{p=1}^{m} P_p \cup \bigcup_{q=1}^{n} P_q^{'}$ denotes the union of permissions in all policies in $Q^{'}$). In this way, $\epsilon^{'}$ satisfies e_i as no less than k_i in U_i together have P_i, and note that adding new users will not lead to violation of policies in $Q^{'}$. For each $f_j \in Q/Q^{'}$, where $f_j = ap\left\langle P_j^{'}, U_j^{'}, t_j\right\rangle$, add all users in $U_j^{'}$ to ϵ, and assign all permissions in $P_j^{'}$ to each of $U_j^{'} \cap S^{'}$ ($S^{'} = \bigcup_{p=1}^{m} U_p \cup \bigcup_{q=1}^{n} U_q^{'}$ denotes the union of users in all policies in $Q^{'}$). In this way, $\epsilon^{'}$ satisfies f_j as there exists one user less than t_j in $U_j^{'}$ together have all permissions in $P_j^{'}$, and note that adding new users, and assigning permissions to these new users will not lead to violation of policies in $Q^{'}$. Therefore, Q is consistent. □

3. Computational Complexities of CCP

The following theorem summarizes the computational complexity results for CCP and its various subcases.

Theorem 4. *The computational complexities of the static consistency checking problem are as follows.*

- *$CCP\langle 1, 1\rangle$ is in P, where $CCP\langle 1, 1\rangle$ denotes the subcase of CCP for which there is a single SSoD policy, and a single availability policy.*
- *$CCP\langle 1, n\rangle$ is coNP-hard, where $CCP\langle 1, n\rangle$ denotes the subcase of CCP for which there is a single SSoD policy, and an arbitrary number of availability policies.*
- *$CCP\langle m, 1\rangle$ is NP-hard, where $CCP\langle m, 1\rangle$ denotes the subcase of CCP for which there is an arbitrary number of SSoD policies, and a single availability policy.*
- *$CCP\langle m, n\rangle$ is in NP^{NP}, where $CCP\langle m, n\rangle$ denotes the most general case of CCP for which there is an arbitrary number of SSoD and availability policies.*

Proof. The proof of Theorem 4 consists of four parts. The first part is Lemma 1, which shows that $\text{CCP}\langle 1, 1\rangle$ can be solved in linear time. In the second part, Lemma 2 shows that $\text{CCP}\langle 1, n\rangle$ is coNP-hard. In the third part, Lemma 4 shows that $\text{CCP}\langle m, 1\rangle$ is NP-hard. In the last part, Lemma 6 shows that $\text{CCP}\langle m, n\rangle$ is in NP^{NP}. □

Lemma 1. *$CCP\langle 1, 1\rangle$ can be solved in linear time.*

Proof. Given a single SSoD policy e=$ssod\langle P, U, k\rangle$, and a single availability policy f=$ap\left\langle P^{'}, U^{'}, t\right\rangle$. According to Theorem 3, we consider only the case where $P \cap P' \neq \emptyset \wedge U \cap U' \neq \emptyset$, and create three subcases to check whether $Q = \{e, f\}$ is consistent as follows:

The first subcase is that $U^{'} \nsubseteq U$: we can construct a state ϵ that assign all permissions in $P^{'}$ to u, and do not assign any permission in P to the users in U. Thus, ϵ satisfies both e and f, Q is consistent.

The second subcase is that $P \nsubseteq P^{'}$, then we can construct a state ϵ as follows: for each $p \in P \wedge p \notin P^{'}$ that, do not assign p to any user in U, and assign all permissions in $P^{'}$ to each user in $U^{'}$. Thus, ϵ satisfies both e and f, and Q is also consistent.

The third case is that $U^{'} \subseteq U \wedge P \subseteq P^{'}$: let $e^{'} = ssod\left\langle P, U^{'}, k\right\rangle$, $f^{'} = ap\left\langle P, U^{'}, k\right\rangle$ and $Q^{'} = \{e^{'}, f^{'}\}$. Q is

consistent if and only if $Q^{'}$ is consistent. For the "only if" part, assume that state ϵ satisfies both e and $f.safe_e(\epsilon)$ being true implies that there exist no less than k users in U together have all permissions in P. It is clear that no less than k users in $U^{'}$ together have all permissions in P as $U^{'} \subseteq U$. In other words, $safe_{e'}(\epsilon)$ is also true. $safe_f(\epsilon)$ being true implies that there is no more than t users in $U^{'}$ together have all permissions in $P^{'}$. Then these t users together can also have all permissions in P as $U^{'}$. In other words, $safe_{f'}(\epsilon)$ is also true. Therefore, if F is consistent, then $Q^{'}$ is consistent. For the "if" part, assume that state ϵ satisfies both $e^{'}$ and $f^{'}$ that both $safe_{e'}(\epsilon)$ and $safe_{f'}(\epsilon)$ are true. We then construct a new access control state ϵ as follows: Firstly, add all the users in $U \cap U^{'}$ to ϵ, but do not assign any permission in P to them. Secondly, add all the permissions in $P \cap P^{'}$ to ϵ, and assign all permissions in $P \cap P^{'}$ to each user in $U^{'}$. $e = ssod\langle P, U, k\rangle$ can be written as $ssod\left\langle P, U^{'} \cup \{U \cap U^{'}\}, k\right\rangle$ as there is no permissions in P being assigned to any user in $U \cap U^{'}$, then there exist no less than k users in U that together have all permissions in P. Thus, $safe_e(\epsilon)$ is true. Similarly, $safe_e(\epsilon)$ is also true. Therefore, if $Q^{'}$ is consistent, then Q is consistent. $e^{'}$ requires that no less than k users in $U^{'}$ together have all permissions in P, and $f^{'}$ requires that no more than t users in $U^{'}$ together have all permissions in P. These two requirements are in conflict if and only if $k > t$, thus $F^{'}$ is consistent when $k \leq t$. Therefore, Q is consistent.

Together with the above discussions, we now give a linear-time algorithm for checking whether $Q = \{e, f\}$ is consistent as follows: Q is consistent if and only if $(U^{'} \nsubseteq U \wedge P \nsubseteq P^{'}) \vee (k \leq t)$; otherwise, Q is inconsistent. □

Lemma 2. *$CCP\langle 1, n\rangle$ is coNP-hard.*

Proof. We reduce the NP-complete set covering problem to $\text{CCP}\langle 1, n\rangle$. In the set covering problem, the input is a finite set S, n subsets of S: $S_1, \cdots, S_n$, and a budget B, the goal is to determine whether the union of B subsets is the same as S. The reduction is as follows. We construct an SSoD policy $e = ssod\langle P, U, B + 1\rangle$, and B availability policies $f_i = ap\langle P_i, U_i, 1\rangle$ $(1 \leq i \leq B)$, where $P = \{p_1, \cdots, p_m\}$ corresponds to S and P_i corresponds to S_i. Let $Q = \{e, f_1, \cdots, f_n\}$, we prove that F is inconsistent if and only if the answer to the set covering problem is "yes" as follows (Lemma 3 asserts that only the availability policy $f_i = ap\langle P_i, U_i, t_i\rangle$ affects the consistency of Q where $P_i \subseteq P \wedge U_i \subseteq U$.

For the "only if" part, we show that if Q is inconsistent, then the answer to the set covering problem is "yes". If a state satisfies each f_i in Q, then e cannot be satisfied such that there exist no more than B users in the state who together have all permissions in P. Let ϵ be a state with n users $u_1, \cdots, u_n$ such that $u_i \in U_i$, and assign all permissions in P_i to u_i. Obviously, ϵ satisfies each availability policy f_i in Q. But there exist no more than B users that together have all permissions in P. Thus, the answer to the set covering problem is "yes" based on above reduction.

For the "if" part, we show that if the answer to the set covering problem is "yes", then Q is inconsistent. If there exist no more than B elements in $\{P_1, \cdots, P_n\}$ whose union is P, let ϵ be a state that satisfies each f_i in Q. For each f_i, let $u_i \in U_i$ and assign all permissions in P_i to u_i. In this case, there are no more than B users in U who together have all the permissions in P. Thus, ϵ does not satisfy the SSoD policy e, which implies that no state satisfies all policies in Q. In other words, Q is inconsistent. □

Lemma 3. *$Q = \{e, f_1, \cdots, f_n\}$, where $e = ssod\langle P, U, k\rangle$, and $f_i = ap\langle P_i, U_i, t_i\rangle$ $(1 \leq i \leq n)$. If $U_i \nsubseteq U$, let $Q^{'} = Q - \{f_i\}$, and if $P_i \nsubseteq P$, let $f_i^{'} = ap\langle P_i \cap P, U_i, t_i\rangle$, $Q^{'} = (Q - \{f_i\}) \cup \{f_i^{'}\}$. Q is consistent if and only if $Q^{'}$ is consistent.*

Proof. If $U_i \nsubseteq U$, there must exist a user $u \in U_i \wedge u \notin U$, and then all the permissions in P_i can be assigned to u, thus f_i can be satisfied, and it does not conflict with e. In other words, the policy f_i where $U_i \nsubseteq U$ does not affect the consistency of Q; therefore, we can delete f_i in Q which makes $Q^{'} = Q$-$\{f_i\}$. As the above discussion, we consider only the case where $U_i \subseteq U$. If $P_i \nsubseteq P$, for each permission $p \in P_i \wedge p \notin P$ that we do not assign p to any user in U_i, and it does not violate the safety requirement of e and the availability requirement of f_i. Therefore, we need not to consider p, and let $f_i^{'} = ap\langle P_i \cap P, U_i, t_i\rangle$, $Q^{'} = (Q - \{f_i\}) \cup \{f_i^{'}\}$. Therefore, Q is consistent if and only if $Q^{'}$ is consistent. □

Lemma 4. *$CCP\langle m, 1\rangle$ is NP-hard.*

Proof. We reduce the NP-complete set splitting problem [8] to $\text{CCP}\langle m, 1\rangle$. In the set splitting problem, the input are a finite set $S = \{s_1, \cdots, s_m\}$, m subsets of S : $S_1, \cdots, S_m$, and we need to determine whether there exist Q_1 and Q_2 such that $Q_1 \cup Q_2 = S$ and there does not exist S_i $(1 \leq i \leq m)$ such that $S_i \subseteq Q_1$ or $S_i \subseteq Q_2$. The reduction is as follows. We construct an availability policy $f = ap\langle P, U, 2\rangle$, and m SSoD policies $e_i = ssod\langle P_i, U_i, 2\rangle$ $(1 \leq i \leq m)$, where $P = \{p_1, \cdots, p_m\}$ corresponds to S and P_i corresponds to S_i. Let $Q = \{e_1, \cdots, e_m, f\}$. We prove that Q is consistent if and only if the answer to the set splitting problem is "yes" as follows (Lemma 5 asserts that only the SSoD policy $e_i = ssod\langle P_i, U_i, k_i\rangle$ affects the consistency of Q where $P_i \subseteq P \wedge U_i \subseteq U$.).

For the "only if" part, we show that if Q is consistent, then the answer to the set splitting problem is "yes". Let a state ϵ satisfy all policies in Q. ϵ satisfying f implies that there exist two users $u_1 \in U$ and $u_2 \in U$ in ϵ such that u_1 and u_2 together have all permissions in P. Furthermore, ϵ satisfying e_i implies that neither u_1 nor u_2 has all permissions in P_i. Let $R_1 = \{s_i | (u_1, p_i) \in UP\}$ and $R_2 = \{s_i | (u_2, p_i) \in UP\}$. Then $R_1 \cup R_2 = S$ and there does not exist S_i $(1 \leq i \leq m)$ such that $S_i \subseteq R_1$ or $S_i \subseteq R_2$. Thus, the answer to the set splitting problem is "yes".

For the "if" part, we show that if the answer to the set splitting problem is "yes", then Q is consistent. Assuming that R_1 and R_2 exist. We construct a state ϵ containing

only two usersu_1 and u_2 such that $auth\_p_\epsilon(u_i) = \{p_j | (p_j \in R_i\}$ $(1 \le i \le 2)$. Then u_1 and u_2 together have all permissions in P as $R_1 \cup R_2$=S which implies that ϵ satisfies f. Since there does not exist S_i $(1 \le i \le m)$ such that $S_i \subseteq R_1$ or $S_i \subseteq R_2$, neither u_1 nor u_2 has all permissions in P_i, which implies that ϵ satisfies e_i. Therefore, ϵ satisfies all policies in Q, in other words, Q is consistent. □

Lemma 5. *$Q = \{f, e_1, \cdots, e_n\}$, where $f = ap\langle P, U, t\rangle$, and $e_i = ssod\langle P_i, U_i, k_i\rangle$ $(1 \le i \le n)$. If $P_i \nsubseteq P$, let $Q' = Q - \{e_i\}$, and if $U_i \nsubseteq U$, let $e_i' = ssod\langle P_i, U_i \cap U, k_i\rangle$, $Q' = (Q - \{e_i\}) \cup \{e_i'\}$. Q is consistent if and only if Q' is consistent.*

Proof. If $P_i \nsubseteq P$, there must exist a user $p \in P_i \wedge p \notin P$, and then does not assign p to any user in U_i that ensures e_i be satisfied and does not affect the satisfaction of f. In other aspect, if $U_i \nsubseteq U$ (we only consider the cases in which $P_i \subseteq P$), we do not assign any permissions in P_i to each user uwhere $u \in U_i \wedge u \notin U$, and it does not violate the safety requirement of e_i and the availability requirement of f. Therefore, we need to consider only the cases where $P_i \subseteq P \wedge U_i \subseteq U$. □

Lemma 6. *$CCP\langle m, n\rangle$ is* $\mathrm{NP}^{\mathrm{NP}}$.

Proof. Let $Q = \{e_1, \cdots, e_m, f_1, \cdots, f_n\}$, where $e_i = ssod\langle P_i, U_i, k_i\rangle$ $(1 \le i \le m)$, $f_j = ap\left\langle P_j', U_j', t_j'\right\rangle$ $(1 \le j \le n)$. We construct a nondeterministic Oracle Turing machine [9] M that makes use of an NP oracle machine to determine whether Q is consistent. The access control state can have all users in U=$(\bigcup_{i=1}^{m} U_i) \cup (\bigcup_{j=1}^{n} U_j')$, in order to satisfy all SSoD policies in Q, the number of users must not be less than $max(k_1, \cdots, k_m)$, and in order to satisfy all availability policies in Q, the number of users must not be more than $\sum_{j=1}^{n} \left|P_j'\right|$. Let M first nondeterministically selects an integer π such that $max(k_1, \cdots, k_m) \le \pi \le \sum_{j=1}^{n} \left|P_j'\right|$ and then generates a size-π user set $U' \subseteq U$. Then M constructs a state ϵ by nondeterminnistically assigning a subset of $\bigcup_{j=1}^{n} P_j'$ to $u \in U$. Then M nondeterministically constructs n sets $U_1', \cdots, U_n'$ of users in ϵ. And then, checking whether users in U_j' $(1 \le j \le n)$ together have all permissions in P_j' and $\left|U_j'\right| \le t_j'$. If the answer is "no", M is rejected; Next, M invokes the NP oracle to check whether ϵ violates any SSoD policies in Q (Theorem 1 shows that checking whether a state violates a set of SSoD policies is coNP-complete, similar to the proof of Theorem 1, it is easy to prove that checking whether a state violates an SSoD policy is in NP). If the oracle machine answers "yes", M is rejected; otherwise, M is accepted, because a state ϵ is found that satisfies all the policies in Q, and hence Q is consistent. Therefore, $\mathrm{CCP}\langle m, n\rangle$ is $\mathrm{NP}^{\mathrm{NP}}$. □

4. An Algorithm for CCP

The fact that $\mathrm{CCP}\langle m, n\rangle$ is intractable ($\mathrm{NP}^{\mathrm{NP}}$) means that there exist difficult problem instances that take exponential time in the worst case. Many instances that will be encountered in practice may still be efficiently solvable. For example, efficient algorithms for $\mathrm{CCP}\langle m, n\rangle$ exist when m and n are small. We now design an algorithm for CCP, which is described in details in this section. Our goal here is to verify that CCP can be solved in reasonable amount of time for general cases, even though the problem is $\mathrm{NP}^{\mathrm{NP}}$ in general. We first give a straightforward algorithm as follows, which is based on the idea of Lemma 6. We then employ a number of improvements based on the straightforward algorithm.

Straightforward Algorithm: Straightforward Algorithm Given a set E of SSoD policies and a set F of availability policies, where E=$\{e_1, \cdots, e_m\}$, F=$\{f_1, \cdots, f_n\}$, where e_i=$ssod\langle P_i, U_i, k_i\rangle$ $(1 \le i \le m)$, f_j=$ap\left\langle P_j', U_j', t_j'\right\rangle$ $(1 \le j \le n)$. Let P=$(\bigcup_{i=1}^{m} P_i \cup (\bigcup_{j=1}^{n} P_j')$, U=$(\bigcup_{i=1}^{m} U_i \cup (\bigcup_{j=1}^{n} U_j')$, for each $u_i \in U$, and each $p_i \in P$, we have a propositional variable $v_{(i,j)}$. This variable is true if p_i is assigned to u_i. All of the variables compose an access control state, and there are $2^{(|P|*|U|)}$ such states. A straightforward algorithm is to enumerate all possible states, and for each such state, determine that whether it satisfies all SSoD policies in E, and all availability policies in F.

Our improved algorithm is based on this idea but we add the following improvements that greatly reduce the running time.

- We perform static pruning to reduce the number of SSoD and availability policies that need to be considered based on Theorem 3, Lemma 7 and Lemma 8.
- We preprocess the input and reduce the number of access control states we need to consider.
- We translate SCP and ACP into two SAT instances, which enable us to benefit from the extensive research on SAT and to use existing SAT solver.

Static Pruning: In the following, we describe a static pruning technique that aims at reducing the number of policies that need to be taken into account. By Theorem 3, given a policy set $Q = \{e_1, \cdots, e_m, f_1, \cdots, f_n\}$, where e_i=$ssod\langle P_i, U_i, k_i\rangle$ $(1 \le i \le m)$, f_j=$ap\left\langle P_j', U_j', t_j\right\rangle$ $(1 \le j \le n)$. The following policies need not be considered, as these policies do not affect the consistency of Q. $\forall e_i \in Q(P_i - R \ne \emptyset)$ and $\forall f_j \in Q(U_j' - S \ne \emptyset)$, where R=$\bigcup_{x=1, x\ne i}^{m} P_x \cup \bigcup_{y=1}^{n} P_y'$, and S=$\bigcup_{x=1}^{m} U_x \cup \bigcup_{y=1, y\ne j}^{n} U_y'$.

Furthermore, we observe that many policies need not be considered either. There is a partial order relation among these policies such that an SSoD policy e_1 is more restrictive than another SSoD policy e_2, or an availability policy f_1 is more restrictive than another availability policy f_2. Then we need not to consider e_2 and f_2. We now explain this pruning technique.

Definition 6. *Let e_1 and e_2 be two SSoD policies, f_1 and f_2 be two availability policies. We say that e_1 is at least as restrictive as e_2(denoted by $e_1 \succeq e_2$) if $\forall\epsilon(safe_{e_1}(\epsilon) \Rightarrow safe_{e_2}(\epsilon))$. We say that f_1 is at least as restrictive as f_2(denoted by $f_1 \succeq f_2$) if $\forall\epsilon(sat_{f_1}(\epsilon) \Rightarrow sat_{f_2}(\epsilon))$.*

The $\geq$ relation among SSoD policies or availability policies is a partial order. When neither $e_1 \geq e_2$ nor $e_2 \geq e_1$, we say e_1 and e_2 are incomparable. In the following, we show how to compare two SSoD policies or availability policies.

Lemma 7. *For any SSoD policies e_1=$ssod\langle P_1, U_1, k_1\rangle$ and e_2=$ssod\langle P_2, U_2, k_2\rangle$, $e_1 \geq e_2$ if and only if $(U_1 \supseteq U_2) \wedge (k_1 \geq k_2 + |P_1 - P_2|)$.*

Proof. For the "if" part, given $(U_1 \supseteq U_2) \wedge (k_1 \geq k_2 + |P_1 - P_2|)$, we show that $\forall\epsilon(\neg safe_{e_2}(\epsilon) \Rightarrow \neg safe_{e_1}(\epsilon))$. There are two cases for $(U_1 \supseteq U_2) \wedge (k_1 \geq k_2 + |P_1 - P_2|)$: (1)$P_1 \subseteq P_2$, (2)$P_1 \supset P_2$. $\neg safe_{e_2}(\epsilon)$ being true means that there exist k_2-1 users in U_2 together have all the permissions in P_2. For case (1), there also exists k_2-1 users in U_1 together have all the permissions in P_1 as $(P_1 \subseteq P_2) \wedge (U_1 \supseteq U_2)$, and $(k_1 \geq k_2 + |P_1 - P_2|) \Rightarrow (k_1 - 1) \geq (k_2 - 1)$. Therefore, there exists k_1-1 users in U_1 together have all the permissions in P_1, in other words, $\neg safe_{e_1}(\epsilon)$ is true, and $\forall\epsilon(\neg safe_{e_2}(\epsilon) \Rightarrow \neg safe_{e_1}(\epsilon))$ is true. For case (2), there also exist k_2-1 users in U_1 together have all the permissions in $P_1 \cup \{P_2 - P_1\}$ as $(U_1 \supseteq U_2)$. At most $|P_1 - P_2|$ users together have all the permissions in $\{P_2 - P_1\}$, and $(k_1 \geq k_2 + |P_1 - P_2|) \Rightarrow (k_2 - 1) \leq (k_1 - 1) - |P_1 - P_2|$. Consequently, there exists k_1-1 users in U_1 together have all the permissions in P_1, $safe_{e_1}(\epsilon)$ is also false.

For the "only if" part, given $e_1 \geq e_2$, we show that $(U_1 \supseteq U_2) \wedge (k_1 \geq k_2 + |P_1 - P_2|)$ is true. Suppose, for the sake of contradiction, that $\neg((U_1 \supseteq U_2) \wedge (k_1 \geq k_2 + |P_1 - P_2|))$ is true. In other words, both $U_1 \supseteq U_2$ and $k_1 \geq k_2 + |P_1 - P_2|$ are false. Let e_1 and e_2 are two SSoD policies, where e_1=$ssod\langle P_1, U_1, k_1\rangle$, e_2=$ssod\langle P_2, U_2, k_2\rangle$. If $U_1 \supseteq U_2$ is false, then $\exists u \in U_2/U_1$, assuming that $safe_{e_1}(\epsilon)$ is true, but assign all the permissions in P_2 to u, then $safe_{e_2}(\epsilon)$ is false as $k_2 > 1$. Therefore, $U_1 \supseteq U_2$ is true. If $k_1 \geq k_2 + |P_1 - P_2|$ is false, then $k_1 < k_2 + |P_1 - P_2|$. If $P_1 \subseteq P_2$, then $k_1 < k_2 \Rightarrow k_1 \leq k_2 - 1$. $safe_{e_1}(\epsilon)$ being true means that at least k_1 users in U_1 together have all the permissions in P_1. We assume that there exist k_1 users in U_1 together having all the permissions in P_1 in ϵ, then there exist k_2-1 users in U_2 together have all the permissions in P_2 as to ϵ (let U_1=U_2, and these k_1 users also have all the permissions in $\{P_2 - P_1\}$), then $safe_{e_2}(\epsilon)$ is false. If $P_1 \supset P_2$, let $k_1 < k_2 + |P_1 - P_2|$, given an access control state ϵ that $safe_{e_1}(\epsilon)$ is true, for each permission in $\{P_2 - P_1\}$, assign it to $|P_1 - P_2|$ different users, and these users are not assigned any other permissions in P_1, then k_1-$|P_1 - P_2|$ users together have all the permissions in P_1. Therefore, there exist less than k_2 users in U_2 together have all the permissions in P_2 (let U_1=U_2), therefore, $safe_{e_2}(\epsilon)$ is false. This contradicts the assumption that $e_1 \geq e_2$. □

Lemma 8. *For any SSoD policies f_1=$ap\langle P_1, U_1, t_1\rangle$ and f_2=$ap\langle P_2, U_2, t_2\rangle$, $f_1 \geq f_2$ if and only if $(P_1 \supseteq P_2) \wedge (U_1 \subseteq U_2) \wedge (t_1 \leq t_2)$.*

Proof. For the "if" part, given $(P_1 \supseteq P_2) \wedge (U_1 \subseteq U_2) \wedge (t_1 \leq t_2)$, we show that $\forall\epsilon(sat_{f_1}(\epsilon) \Rightarrow sat_{f_2}(\epsilon))$ is true. $sat_{f_1}(\epsilon)$ being true means that there exists no more than t_1 users in U_1 together have all the permissions in P_1. Since $(P_1 \supseteq P_2) \wedge (U_1 \subseteq U_2) \wedge (t_1 \leq t_2)$, then there also exists no more than t_2 users in U_2 together have all the permissions in P_2, in other words, $sat_{f_2}(\epsilon)$ is also true.

For the "only if" part, given $f_1 \geq f_2$, we show that $(P_1 \supseteq P_2) \wedge (U_1 \subseteq U_2) \wedge (t_1 \leq t_2)$ is true. Suppose, for the sake of contradiction, that $\neg(P_1 \supseteq P_2) \wedge (U_1 \subseteq U_2) \wedge (t_1 \leq t_2)$ is true, thus $\neg(P_1 \subset P_2) \vee (U_1 \supset U_2) \vee (t_1 > t_2)$ is true. Let f_1 and f_2 are two availability policies, where f_1=$ap\langle P_1, U_1, t_1\rangle$, f_2=$ap\langle P_2, U_2, t_2\rangle$. If $P_1 \subset P_2$ is true, then $\exists p \in P_2/P_1$. Assuming that there exists an access control state ϵ, and $sat_{f_1}(\epsilon)$ is true. Let p not be assigned to any user in U_2, that does not affect $sat_{f_1}(\epsilon)$. But $sat_{f_2}(\epsilon)$ is false, because no t_2 users together have all the permissions in P_2. This contradicts the assumption that $f_1 \geq f_2$; therefore, $P_1 \subset P_2$ is false, and $P_1 \supseteq P_2$ is true. If $U_1 \supset U_2$ is true, then $\exists u \in U_1/U_2$. We now construct a state ϵ by assigning all the permissions in P_1 to u, and do not assign any permissions in P_1 to any other users in U_1. That makes $sat_{f_1}(\epsilon)$ is true as u has all the permissions, but $sat_{f_1}(\epsilon)$ is false, because there exists no *size-t_2* user set covering all the permissions in $P_2(P_1 \supseteq P_2)$. This contradicts the assumption that $f_1 \geq f_2$, therefore, $U_1 \supset U_2$ is false, and $U_1 \subseteq U_2$ is true. If $t_1 > t_2$ is true, we now construct a state ϵ that t_1 users in U_1 together have all the permissions in P_1, but do not assign any other permissions to any user in $U_2 - U_1$, that makes $sat_{f_1}(\epsilon)$ is true, but $sat_{f_2}(\epsilon)$ is false, because there does not exist no more than t_2 users in U_2 together having all the permissions in P_2 $(t_1 > t_2)$. This contradicts the assumption that $f_1 \geq f_2$; therefore, $t_1 > t_2$ is false, and $t_1 \leq t_2$ is true. Consequently, if $f_1 \geq f_2$, then $(P_1 \supseteq P_2) \wedge (U_1 \subseteq U_2) \wedge (t_1 \leq t_2)$. □

Preprocessing: Theorem 2 shows that determining whether an access control state ϵ satisfies a set F of availability policies is NP-complete. But it is also easy to eliminate many states that do not satisfy all the availability policies in F. Given a state ϵ, and a availability policy set F, for each $f_i \in F$, where f_i=$ap\langle P_i, U_i, t_i\rangle$, if $\bigcup_{u \in U_i} Perm(u) \not\supseteq P_i$, then we know that the state does not satisfy f_i, and ϵ needs not be considered.

Reduction to SAT: After employing preprocessing and static pruning techniques to reduce the number of access control states, SSoD policies and availability policies that need to be considered, the next key step to solve CCP is to determine whether a given access control state satisfies a set of SSoD and availability policies. Let e_i=$ssod\langle P_i, U_i, k_i\rangle$, f_j= $ap\left\langle P_j^{'}, U_j^{'}, t_j\right\rangle$, given a state ϵ, $safe_{e_i}(\epsilon)$ being true means that there does not exist less than k_i users in U_i together have all the permissions in P_i, and $sat_{f_j}(\epsilon)$ being true means that there exists no more than t_j users in $U_j^{'}$ together having all the permissions in $P_j^{'}$. Obviously, SSoD policies are a natural complement to availability policies. If there exist no more than k_i-1 users in U_i together having all the permissions in P_i, then $safe_{e_i}(\epsilon)$ is false, but $sat_{f_i}(\epsilon)$ is true, where f_i= $ap\langle P_i, U_i, k_i - 1\rangle$. Consequently, we regard SCP

as a reverse problem of ACP, and observe that ACP can be translated into a SAT instance. In this way, we can use algorithms for SAT to solve ACP. The SAT solver we use is SAT4J [10], and it supports Pseudo-Boolean constraints, which are linear inequalities with integer coefficients.

The translation works as follows. Given an availability policy $f{=}ap\langle P, U, t\rangle$, for each $u_i \in U$, we have a propositional variable v_i. This variable is true if u_i is a member of *size-t* user set $U' \subseteq U$ to cover all the permissions in P. Then we have the following two kinds of constraints. For each $p \in P$, let $u_{i_1}, u_{i_2}, \cdots, u_{i_x}$ be the users who are authorized for the permission p. We add the first constraint $v_{i_1} + v_{i_2} + \cdots, v_{i_x} \geq 1$, which ensures that all the permissions in P are covered by U'. There are m ($m = |P|$) such constraints. Then we add the second constraint $v_{i_1} + v_{i_2} + \cdots, v_{i_x} \leq t(n = |U|)$, which ensures that $\left|U'\right| \leq t$. There is only one such constraint.

5. Evaluation and Illustration

5.1 Implementation and Evaluation

We prototyped the algorithm described in Sect. 4 and have performed some experiments using randomly generated instances. Our prototype is written in Java, we use SAT4J, an open source satisfiability library in Java. The Pseudo Boolean constraints used in our prototypes are supported by SAT4J. The experiments were carried out on a machine with an Intel(R) Core(TM)2 Duo CPU T5750 running at 2.0 GHz, and with 2 GB of RAM running Microsoft Windows XP Professional.

The methodology that we use in generating test instances of SSoD and availability policies is as follows. In practice, the number of permissions involved in an access control system will not be very large. However, the number of users in the system may be large. In this way, we assume that there are 20 permissions and 50 users in an access control system. The number of permissions involved in an SSoD policy will not be very large, and the number of users in the policy may be large. In order to approximate realistic instances, we generate an SSoD policy $e{=}ssod\langle P, U, k\rangle$ for testing using combinations of the following approaches. The generation of an availability policy $f{=}ap\langle P, U, t\rangle$ is essentially the same as follows.

- $|P|{=}X$, where X is a random integer variable, $2 \leq X \leq 20$, with a distribution of the density function $f(x){=}\frac{e^{-(x-4)^2}}{\sqrt{2\pi}}$, we write X˜N(4,1).
- $|U|{=}Y$, where Y is a random integer variable, $2 \leq Y \leq 50$, with a distribution of the density function $f(y){=}\frac{e^{-(y-5)^2}}{\sqrt{2\pi}}$, we write Y˜N(5,1).
- k is a random integer in $[2, min(|P|, |U|)]$.
- When $|P|$ and $|U|$ are generated, then randomly select $|P|$ integers $\{i_1, \cdots, i_x\}$ from [2, 20] to be the right suffix for the permissions $\{p_{i_1}, \cdots, p_{i_x}\}{=}P$. Also randomly select $|U|$ integers $\{j_1, \cdots, j_y\}$ from [2, 50] to be the right suffix for the users $\{u_{j_1}, \cdots, u_{j_y}\}{=}U$.

Static pruning is very effective: Table 1 shows the effect of static pruning for increasing values of m (number of SSoD policies) and n (number of availability policies). The number of SSoD and availability policies after static pruning is fewer, for example, for $m = 2$, and $n = 2$, 4, or 6, there is no SSoD or availability policies that need to be checked with static pruning. The SSoD and availability policies be decreased by about 68% through static pruning, where the total number of policies is 224, and the number of remainders is 71. Furthermore, the basic idea of static pruning is to reduce the number of policies that need to be taken into account by Theorem 3, Lemma 7 and Lemma 8. Generally, the SSoD policies with many permissions will not be considered, and the availability policies with many users will not be considered too.

Preprocessing is also effective: Table 2 shows the effect of preprocessing for increasing values of m (number of SSoD policies) and n (number of availability policies). The access control states that need to be considered will be decreased by about 20%˜30% through static pruning. Obvi-

Table 1 A table that shows static pruning is effective (the number of SSoD and availability policies should be considered). The columns are values for n (number of availability policies) and rows are values for m (number of SSoD policies). For each cell in the table, the entry above the dotted line is the number of SSoD and availability policies that need to be checked with static pruning in effect, and the number below the dotted line is the number of SSoD and availability policies that need to be checked without static pruning.

m \ n	2	4	6	8	10
2	0 / 4	0 / 6	0 / 8	2 / 10	2 / 12
4	0 / 6	2 / 8	2 / 10	3 / 12	4 / 14
6	2 / 8	2 / 10	3 / 12	4 / 14	4 / 16
8	2 / 10	3 / 12	3 / 14	4 / 16	4 / 18
10	4 / 12	4 / 14	5 / 16	5 / 18	6 / 20

Table 2 A table that shows preprocessing is effective (the number of access control states should be considered). The columns are values for n (number of availability policies) and rows are values for m (number of SSoD policies). For each cell in the table, the entry above the dotted line is the number of access control states that need not to be considered with preprocessing in effect, and the number below the dotted line is the number of access control states that need not to be considered without preprocessing.

m \ n	2	4	6	8	10
2	0 / 0	0 / 0	0 / 0	73 / 256	127 / 512
4	0 / 0	79 / 256	149 / 512	430 / 2048	3932 / 16384
6	37 / 128	317 / 1024	2129 / 8192	7208 / 32768	12451 / 65536
8	69 / 256	471 / 2048	2213 / 16384	23593 / 131072	141557 / 524288
10	1269 / 4096	1802 / 8192	15834 / 65536	136314 / 524288	325058 / 1048576

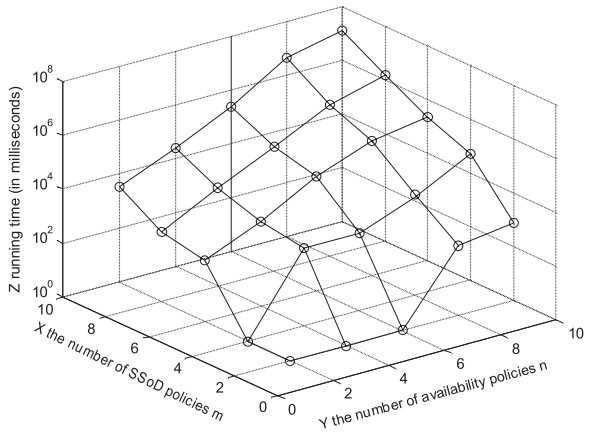

Fig. 1 This graph shows the effect on running time (in milliseconds) as the number of SSoD policies m and the number of availability policies n increase.

ously, the preprocessing does scale well with the number of SSoD and availability policies grows.

The algorithm scales reasonably well with m and n when m and n are not very large: Our experimental results are presented in Fig. 1. The running time of our algorithm depends on the number of SSoD and availability policies, the total number of access control states which are need to be considered, and the time spent in the SAT solver, which is greatly influenced by the number of permissions and users in an SSoD or availability policy. Our improved algorithm is able to solve CCP$\langle m, n\rangle$ in reasonable amount of time when m and n are not very large. For example, for $m = 6$ and $n = 4$, the algorithm takes only about 1.4 seconds. However, as m and n become larger, the algorithm stops scaling. For example, for $m = 10$ and $n = 10$, the algorithm takes about 3 hours, which is unacceptable. In particular, the algorithm takes less than 1 second, where the number of SSoD and availability policies that need to be checked is only 2. Via Lemma 1, we know that given two policies e=$ssod\,\langle P, U, k\rangle$ and f=$ssod\left\langle P', U', t\right\rangle$, let Q={e, f}, Q is consistent if and only if$(U' \not\subseteq U \vee P \not\subseteq P') \vee (k \leq t)$; otherwise,$Q$ is inconsistent. In this way, we need not spent any time in the SAT solver, which can greatly improving the performance of our algorithm. However, as m and n becomes larger, the number of the access control states need to be considered will increase enormously. Consequently, lots of time will be spent in the SAT solver. As CCP needs to be performed only when the access control state of the system changes, which is not expected to happen frequently, relative slow running time may be acceptable in some situations. Further research is needed on improving the performance of the algorithm.

5.2 Illustration and Analysis

We now give a running example to show the validity of our algorithm for consistency checking problem.

Example 1: Consider the task of ordering and paying for goods discussed in Sect. 1. We have a permission corresponding to each step in the task; these permissions are *order*, *goods*, *invoice*, and *payment*. A set E of SSoD policies $\{e_1, e_2, e_3, e_4\}$ and a set F of availability policies $\{f_1, f_2, f_3\}$ exactly meet the safety requirements (i.e., $\{(a), (b), (c), (d)\}$) and the availability requirements (i.e., $\{(e), (f), (g)\}$) respectively. We have the following expressions.

- E={e_1, e_2, e_3, e_4}.
- $e_1 = ssod <$ {*order*, *goods*, *invoice*, *payment*}, {*Alice*, *Bob*, *Carl*, *Doris*}, 3 >.
- e_2=$ssod <$ {*order*, *goods*}, {*Alice*, *Bob*, *Carl*, *Doris*}, 2 >.
- e_3=$ssod <$ {*goods*, *invoice*}, {*Bob*, *Carl*, *Doris*}, 2 >.
- e_4=$ssod <$ {*order*, *goods*, *invoice*}, {*Alice*, *Bob*, *Carl*, *Doris*}, 3 >.
- F={f_1, f_2, f_3}.
- f_1=$ap <$ {*order*, *goods*, *invoice*, *payment*}, {*Alice*, *Bob*, *Carl*}, 3 >.
- f_2=$ap <$ {*order*, *payment*}, {*Alice*, *Bob*}, 1 >.
- f_3=$ap <$ {*order*, *goods*, *payment*}, {*Alice*, *Bob*, *Carl*}, 2 >.

We now implement the proposed algorithm to resolve the consistency checking problem of a set E of SSoD policies and a set F of availability policies. Let Q=$E \cup F$={$e_1, e_2, e_3, e_4, f_1, f_2, f_3$}.

Firstly, we perform static pruning to reduce the number of these policies that need to be considered. Via Theorem 3, $e_1 \in Q(P_1 - R = \{payment\} \neq \emptyset)$, where

- R={*order*, *goods*, *invoice*},
- P_1={*order*, *goods*, *invoice*, *payment*},

Obviously, e_1 does not affect the consistency of Q. Let Q'=$Q - \{e_1\}$. Via Lemma 7, both $(U_4 \supseteq U_2) \wedge k_4 \geq k_2 + |P_4 - P_2|)$ and $(U_4 \supseteq U_3) \wedge k_4 \geq k_3 + |P_4 - P_3|)$ are true, where

- U_2={*Alice*, *Bob*, *Carl*, *Doris*},
- U_3={*Bob*, *Carl*, *Doris*},
- U_4={*Alice*, *Bob*, *Carl*, *Doris*},
- P_2={*order*, *goods*},
- P_3={*goods*, *invoice*},
- P_4={*order*, *goods*, *invoice*},
- k_2=2, k_3=2, k_4=3.

Let Q'= Q'-{e_2, e_3}. Via Lemma 8, $(P'_1 \supseteq P'_2) \wedge (U'_1 \subseteq U'_2) \wedge (t_1 \leq t_2)$, where

- P'_1={*order*, *goods*, *invoice*, *payment*},
- P'_2={*order*, *payment*},
- U'_1={*Alice*, *Bob*, *Carl*},
- U'_2={*Alice*, *Bob*},
- t_1=3, t_2=1.

Then let Q'=Q'-{f_2}. Based on the above discussion, Q={$e_1, e_2, e_3, e_4, f_1, f_2, f_3$} is consistent if and only if Q'={e_4, f_1, f_3} is consistent.

Secondly, a straightforward algorithm is to enumerate

all possible states, and there are 216 such states. We then perform preprocess to reduce the number of access control states that need to be considered. For each $f_i \in \{f_1, f_3\}$, the access control state ϵ needs not to be considered, if $\bigcup_{u \in U_i} Perm(u) \not\supseteq P_i$, where

- $P_1^{'}$={*order*, *goods*, *invoice*, *payment*},
- $P_3^{'}$={*order*, *payment*, *payment*},
- $U_1^{'}$={*Alice*, *Bob*, *Carl*},
- $U_3^{'}$={*Alice*, *Bob*, *Carl*}.

There exist 27120 such access control states that should not be considered. Only 10640 access control states need to be considered, which are decreased by about 71.82%. This shows the preprocessing is effective. Thirdly, we reduce the consistency checking problem to SAT solver. Finally, There exist 72 access control states in which $safe_{e_4}(\epsilon) \wedge sat_{\{f_1,f_3\}}(\epsilon)$ is true. Therefore, the policies in $Q^{'}$ are consistent, thus the set E of SSoD policies and the set F of availability policies are consistent.

6. Related Work

While policy analysis has been the main research area in access control for several decades, almost all previous work in access control analysis focused on safety property, which determines whether an access control system can reach a state where an unsafe access is allowed. Based on the understanding that security analysis is a critical problem for trust management, Li et al. [1] proposed the notion of security analysis, which generalizes safety analysis in the context of a trust management framework. They also studied the security analysis in the context of role-based access control (RBAC), where they gave a precise definition of a family of security analysis problems in RBAC. It is more general than safety analysis that is studied in the literature [11]. Zhang et al. investigate the safety property with pre-authorization policies in usage control (UCON) [12].

One example of security policy is separation-of-duty (SoD) policy, which is a very important principle in information security. The concept of SoD can be traced back to 1975 when Saltzer and Schroeder [13] took it as one of the design principles for protecting information, under the name "separation-of-privilege". Since Clark and Wilson [3] applied SoD principle to the data objects to ensure integrity and control frauds along with well-formed transactions as two major mechanisms for controlling fraud and error. Later on, SoD has been vastly studied by various researchers as a principle to avoid frauds. There exists a wealth of literature on SoD policies in the context of RBAC [14]. It has been recognized that "one of RBAC's great advantages is that SoD rules can be implemented in a natural and efficient way". The specification of SoD policies is a very important issue. It should be noted that most existing approaches on separation-of-duty only consider constraint sets with exact two elements. The distinction between the SSoD policy objectives and the SMER (statically mutually exclusive role) constraints, as a mechanism to enforce them, is sometimes not clearly made [15]. Motivated by the work of Li et al [16], we formally define SSoD policies. Besides, we consider the total number of available users as a limitation factor through referring to the Jason's work [4].

The security policy focusing on safety properties is mostly viewed as a tool for restricting access. An equally important aspect of access control is to enable access. In this way, we introduce the notion of availability policies in this paper, which state properties about enabling access in access control. Li et al introduces the related concept of availability policies in [1], [11], which discriminates whether a user always possesses certain permissions across state changes. Unlike the availability policies in the work by Li et al, the availability policies in this paper state that the cooperation of at most a certain number of users is required to complete a task which ensures that the task does not become stuck and therefore cannot be completed. The availability policy is a high-level requirement, and it is expressed in terms of restrictions on permission set and user set. It does not specify a permission requirement on any individual user. A similar concept is resiliency policy [17], which requires an access control system to be resilient to the absence of users. However, the resiliency policy does not consider the total number of available users as a limitation factor, since the number of users in any organization is bounded in practice. The requirement of resiliency policy is to tolerate absent users and the overall ability of groups of users to perform critical tasks.

Both security policies and availability policies are very important. In many cases, it is desirable for access control system to have both of them. In practice, the security officer may design many security policies and availability policies in an access control system. Due to their opposite focus, these policies may conflict with each other. Consequently, this paper attempts to address the problem of consistency checking for safety and availability in the context of access control, which is very important for security officers, although it may become a challenging task.

7. Conclusions and Future Work

In this paper, we give the formal definition of SSoD policies, and introduce the notion of availability policies. It is desirable for an access control system having both SSoD and availability policies. However, these policies may conflict with each other due to their opposite focuses. This paper addresses the problem of consistency checking for the co-existence of SSoD and availability policies. We show that the general case of the consistency checking problem and several subcases are intractable (NP-hard), and is in the Polynomial Hierarchy (in NP^{NP}). Although the consistency checking problem is intractable in general, many instances that will be encountered in practice may still be efficiently solvable. Therefore, we design an algorithm for solving this problem in general case. The algorithm can efficiently solve instances of nontrivial sizes that belong to the intractable cases of the problem.

In the future research, we intend to address the problem of consistency checking for the co-existence of other policies with conflict focuses. For example, a cardinality constraint focuses on the restriction of the role cardinality of a relationship. A cardinality constraint may specify a minimum or maximum number of related entities. It may conflict with both SSoD and availability policies. This problem seems to be particularly interesting in role-based access control systems. Another open area lies in the design techniques for resolving the inconsistency in optimal way. This paper explores the co-existence of SSoD and availability policies. When these policies are inconsistent, conflicts are thus resolved by withdrawing one or more policies till the conflicts are corrected. Resolution of policy conflicts by manual intervention of policy administrator is a slow and ad hoc process. Thus, an optimal conflict resolution technique is very important.

Acknowledgments

We would like to thank Dr Yong Zeng at Concordia University for his many suggestions that improved the content and presentation of this paper. This work is supported by National Natural Science Foundation of China under Grant 60873225, 60773191 and 70771043, National High Technology Research and Development Program of China under Grant 2007AA01Z403.

References

[1] N. Li, J.C. Mitchell, and W.H. Winsborough, "Beyond proof-of-compliance: Security analysis in trust management," J. ACM, vol.52, no.3, pp.474–514, May 2005.
[2] N. Li and M.V. Tripunitara, "Security analysis in role-based access control," Proc. 9th ACM Symposium on Access Control Models and Technologies (SACMAT), pp.126–135, Yorktown Heights, New York, USA, June 2004.
[3] D.D. Clark and D.R. Wilson, "A comparison of commercial and military computer security policies," Proc. 8th IEEE Symposium on Security and Privacy (SP), IEEE Computer Society Press, pp.184–195, Oakland, California, USA, April 1987.
[4] J. Crampton, "Specifying and enforcing constraints in role-based access control," Proc. 8th ACM Symposium on Access Control Models and Technologies (SACMAT), pp.43–50, Villa Gallia, Como, Italy, June 2003.
[5] J.A. Solworth, "Approvability," Proc 1st ACM Symposium on Information, Computer and Communications Security (ASIACCS), pp.231–241, Taipei, Taiwan, March 2006.
[6] J. Lu, R. Li, Z. Lu, J. Hu, and X. Ma, "Specification and enforcement of static separation-of-duty policies in usage control," Proc. 12th Information Security Conference (ISC), pp.403–410, Pisa, Italy, Sept. 2009.
[7] C.H. Papadimitriou, Computational Complexity, Addison Wesley Longman, 1994.
[8] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, W.H. Freeman, ISBN 0-7167-1045-5, New York, 1979.
[9] N. Li and Q. Wang, "Beyond separation of duty: An algebra for specifying high-level security policies," Proc. 13th ACM Conference on Computer and Communication Security (CCS), pp.356–369, Alexandria, Virginia, USA, Nov. 2006.
[10] D.L. Berre (project leader), SAT4J: A satisfiability library for Java, URL http://www.sat4j.org/, Jan. 2006.
[11] N. Li and M.V. Tripunitara, "Security analysis in role-based access control," Proc. 9th ACM Symposium on Access Control Models and Technologies (SACMAT), pp.126–135, June 2004.
[12] X. Zhang and R. Sandhu, "Safety analysis of usage control authorization models," Proc. 1st ACM Symposium on Information, Computer and Communications Security (ASIACCS), pp.243–254, Taipei, Taiwan, March 2006.
[13] J.H. Saltzer and M.D. Schroeder, "The protection of information in computer systems," Proc. IEEE, vol.63, no.9, pp.1278–1308, Sept. 1975.
[14] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," Computer, vol.29, no.2, pp.38–47, Feb. 1996.
[15] D.F. Ferralolo, J.A. Cuigini, and D.R. Kuhr, "Role-based access control (RBAC): Features and motivations," Proc. 11th Annual Computer Security Applications Conference (ACSAC), pp.12–14, New Orleans, Dec. 1995.
[16] N. Li, M.V. Tripunitara, and Z. Bizri, "On mutually exclusive roles and separation-of-duty," Proc. ACM Transactions on Information and System Security, vol.10, no.2, pp.1–36, May 2007.
[17] N. Li, M.V. Tripunitara, and Q. Wang, "Resiliency policies in access control," Proc. 13th ACM Conference on Computer and Communication Security (CCS), pp.113–123, Alexandria, Virginia, USA, Nov. 2006.

Ruixuan Li received the B.S., M.S., and Ph.D. degrees from College of Computer Science and Technology at Huazhong University of Science and Technology in 1997, 2000, and 2004, respectively. Since 2004, He has been an Associate Professor of College 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.

Jianfeng Lu received the B.S. degree from College of Computer Science and Technology at Wuhan University of Science and Technology in 2005. He is a PhD candidate in the Intelligent and Distributed Computing Lab, College of Computer Science and Technology, Huazhong University of Science and Technology, and is expected to graduate in June 2010. His research interests include access control, policy analysis, separation-of-duty, and secure interoperation.

Zhengding Lu is currently a professor in College of Computer Science and Technology at Huazhong University of Science and Technology, and is the director of the Intelligent and Distributed Computing Laboratory. His research interests include distributed computing, distributed database, distributed system security, and multimedia information systems.

Xiaopu Ma received his M.S. degree in School of Computer Science and Engineering from University of Electronic Science and Technology of China in 2004. Now he is a Ph.D. candidate in the Intelligent and Distributed Computing Lab, College of Computer Science and Technology, Huazhong University of Science and Technology. His research interests include distributed system security, access control, and multidomain interoperation.