Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

An authorization model for collaborative access control*

Chen-hua MA†1, Guo-dong LU[1], Jiong QIU[2]

([1]*Engineering and Computer Graphics Institute, Zhejiang University, Hangzhou 310027, China*)

([2]*Department of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China*)

†E-mail: mchma@zju.edu.cn

Received Sept. 14, 2009; Revision accepted Dec. 7, 2009; Crosschecked Aug. 2, 2010

Abstract: Collaborative access control is receiving growing attention in both military and commercial areas due to an urgent need to protect confidential resources and sensitive tasks. Collaborative access control means that multiple subjects should participate to make access control decisions to prevent fraud or the abuse of rights. Existing approaches to access control cannot satisfy the requirements of collaborative access control. To address this concern, we propose an authorization model for collaborative access control. The central notions of the model are collaborative permission, collaboration constraint, and collaborative authorization policy, which make it possible to define the collaboration among multiple subjects involved in gaining a permission. The implementation architecture of the model is also provided. Furthermore, we present effective conflict detection and resolution methods for maintaining the consistency of collaborative authorization policies.

Key words: Collaborative access control, Collaborative permission, Conflict detection and resolution
doi:10.1631/jzus.C0910564 **Document code:** A **CLC number:** TP309.2

1 Introduction

The activities within a computer system can be viewed as a sequence of operations on objects. One of the primary purposes of a security mechanism is access control that consists of determining and enforcing which active subjects, e.g., processes, can have access to which objects and in which access mode (Koch *et al.*, 2002).

Collaborative access control means that the cooperation of multiple subjects is required in making an access control decision; i.e., an access request can be granted only with the agreement or acceptance of two or more subjects. Requirements for collaborative access control have long existed in the physical world, such as the two-man rule, whereby the cooperation of at least two different users is required to complete a sensitive task. In real-life scenarios, there are several motivations for collaborative access control:

1. The first is to protect the confidentiality and integrity of sensitive resources. For example, when a business involves several partners, it may be necessary to control individual access to sensitive data such as contracts, design details, trade figures, and business strategies, some or all of which may be considered commercial secrets. Malicious or even unintended alteration of this information may not only lead to disastrous internal management decisions but also affect the reputation of an enterprise (Tan *et al.*, 2002). In such a scenario, the cooperation of multiple subjects might be required to grant permission to access these sensitive data. If a subject must access specific sensitive data, then committing fraud requires a conspiracy of at least two people, which raises significantly the probability of disclosure and capture.

2. The second motivation is the performance of sensitive tasks that exist in both commercial and military areas. Decisions concerning authorization to execute such tasks often require approval from mul-

* Project (Nos. 2008C13073 and 2009C03015-1) supported by the Large Science and Technology Special Social Development Program of Zhejiang Province, China

tiple independent parties to prevent fraud and the abuse of rights. For instance, an important commercial investment strategy must be approved by at least half of the members including the chairman of the board of directors. Any individual, including the highest-level individuals, cannot be granted permission to approve such strategies unless the required collaboration among various parties is achieved.

Collaborative access control has several key requirements:

1. Permission weights should be considered to reflect the degree of trust held by different subjects in accessing a permission. In collaborative access control, different subjects may have different degrees of trust in gaining a permission. The degree of trust refers to the level of liability and trustworthiness that a subject has in a collaboration. For example, a general manager might have a higher degree of trust in collaborative access to business strategies than a common employee. A subject with a higher degree of trust in gaining a permission should be assigned a larger weight towards the permission. This is different from regular permission assignments in existing access control approaches.

2. A flexible approach should be followed to define the collaboration among multiple subjects for gaining a permission. In collaborative access control, a permission can be accessed only if the corresponding collaboration is achieved. For example, the permission to read a confidential document can be accessed only if at least three different users including a general manager agree with the access request, and the total weight of the permission owned by these users should be equal to or greater than 6. Different permissions may have different required collaborations. For instance, a permission with a higher degree of sensitivity (i.e., one involving a very sensitive protected resource) may require the involvement of more collaborators.

3. Context constraints (e.g., time, location, and historical information) are an important factor that should be specified in collaborative access control to support the least privilege principle; i.e., every subject should operate using the least amount of privilege necessary to complete a specific job. For example, a common employee can be granted the permission to read confidential documents with a weight of 1 only when access time is between 9 am and 5 pm.

Existing approaches to access control cannot satisfy the above requirements. To address the problem, we propose an authorization model for collaborative access control (CACAM) in the context of RBAC (role-based access control), which is accepted as the access control model most suitable for enterprise organizations. Our contributions are summarized as follows:

1. The concepts of collaborative permission, collaboration constraint, and collaborative authorization policy are proposed to address the above requirements. Collaborative permissions refer to the permissions that need to be accessed collaboratively by multiple parties. Each collaborative permission is associated with a collaboration constraint, which makes it possible to express the conditions that need to be satisfied by collaborators to gain the permission flexibly. Collaborative permissions are assigned to roles via the specification of collaborative authorization policies. An important feature of the policy is that it can specify how much weight a role carries towards attaining a specific collaborative permission.

2. Effective conflict detection rules for collaborative authorization policies are proposed to prevent the occurrence of inconsistent system states. Moreover, we introduce a flexible approach to resolve conflicts by defining two new concepts, priority rule and resolution policy.

3. We present the implementation architecture for CACAM and describe the collaboration among multiple subjects for gaining a collaborative permission.

2 Related works

In the past decade, a considerable amount of work has been done on access control. The research most directly related to ours has been in the areas of access control in collaborative applications, privacy-aware access control, and separation of duty (SoD).

To fulfill the security requirements in collaborative environments, many related studies have been carried out. Traore and Khan (2003) proposed a flexible protection scheme based on a lattice security model that combines information flow and access control mechanisms to address privacy and integrity

requirements in scalable collaborative environments. In their scheme, an owner of a shared resource can attach security policies that define who is allowed to do what operations to the resource, thus preventing the leakage of valuable data to others. Li *et al.* (2007) addressed the issue of access control in a corporate collaboration setting, proposed guidelines for setting up multiple access control policies, and discussed the conflicts that occur between access policies. However, the approaches proposed in these studies did not address the collaboration among multiple subjects in making access control decisions.

Alsulaiman *et al.* (2007) presented a threshold-based collaborative access control model T-CAC, which makes it possible to specify some collaboration among various users to obtain permission. However, T-CAC defines only the weight required by collaborators to gain the requested permission. Other important factors in collaboration, such as the least number of collaborators used to prevent fraud and the abuse of rights, were overlooked. Moreover, T-CAC does not address the specification of context constraints in authorization policies, and the conflict detection and resolution methods used to prevent the occurrence of inconsistent system states have not been addressed.

Carminati and Ferrari (2008) addressed the issue of privacy-aware collaborative access control in Web-based social networks (WBSNs) by presenting a protocol based on a collaboration of selected nodes in the network. Kim *et al.* (2008) proposed a collaborative model based on XACML (extensible access control markup language) of the OASIS (open architecture for securely interworking services) standard in pervasive environments. However, permission weight reflecting the trustworthiness of collaborators was not considered.

Li *et al.* (2008) proposed a novel access control model for a multi-party collaborative environment and developed an extension to the reference architecture for XACML to support collaborative access control. However, the proposed model aims to decompose global policies into local policies and strategies to protect sensitive information belonging to one party from being known by other parties. This model is not suitable for the scenarios discussed above and cannot satisfy the requirements of collaborative access control.

To address the issue of privacy protection, several privacy-related access control models and approaches have been proposed. Franz *et al.* (2006) discussed how protection of data can be realized in a privacy-enhanced, flexible eLearning environment by using anonymous credentials and different access control policies provided by PRIME (privacy and identity management for Europe, a European project aiming at developing a working prototype of a privacy-enhancing identity management system). Ni *et al.* (2007) introduced a family of models (P-RBAC) that extend the well-known RBAC model to provide full support for expressing highly complex privacy-related policies. In P-RBAC, a privacy policy is expressed as a privacy permission, which explicitly states the intended purpose, along with the conditions under which the permission can be given, and the obligations that are to be finally performed. P-RBAC unifies privacy policy enforcement and access control policy enforcement into one access control model. Ardagna *et al.* (2008) presented a privacy-aware framework that integrates access control policies together with a new type of privacy policy, called a data handling policy, which allows users to specify and regulate how personal identifiable information will be handled by the receiving parties. However, these studies did not address the collaboration among multiple users to gain privacy-related permissions.

SoD is a well-known security principle used to formulate multi-person control policies. Its purpose is to prevent fraud and error by spreading the responsibility and authority for an action or task over multiple people, thereby raising the risk involved in committing a fraudulent act by requiring the involvement of more than one individual (Simon and Zurko, 1997). In the past 20 years, a considerable amount of work (Michael *et al.*, 1990; Simon and Zurko, 1997; Gligor *et al.*, 1998; Ahn and Sandhu, 2000; Crampton, 2003; Joshi *et al.*, 2003; Sohr *et al.*, 2005) has been done on SoDs and a variety of SoD variations have been presented (Table 1).

Note that these variations focus on controlling membership in, activation of, and use of roles, and aim at dividing a business process into several steps, spreading the responsibilities and authorities for these steps over multiple people and requiring that each step be performed by a different person. These SoD variations do not provide multi-person control over

Table 1 Major SoD variations proposed in previous studies

No.	SoD variation	Description
1	Conflicting roles (CR)	Two or more roles of a conflicting role set cannot be assigned to the same user
2	Conflicting permissions (CP)	A user cannot acquire conflicting permissions
3	Conflicting users (CU)	Two conflicting users cannot have roles in the same conflicting role set
4	User-based static SoD	No role in a specific set is assigned to two or more users in a specific set
5	Simple dynamic SoD (User-based dynamic SoD)	Users may not assume two restricted roles at the same time. Restricted roles refer to roles that have constraints on their activation or use
6	Session-based SoD	Restricted roles cannot be active at the same time in the same user session
7	Object-based SoD	No user may act upon a target that the user has previously acted upon
8	Operational SoD	No user may assume a set of roles that contain all the actions in a complete business task
9	History-based SoD	No user is allowed to perform all the actions in a business task on the same target or collection of targets
10	Role enabling/disabling time-based SoD	Roles from a given role set cannot be enabled/disabled at the same time in a given time interval
11	Assignment time-based SoD	A user or a permission can be assigned to a role in a specific time interval
12	Activation time-based SoD	Activation time-based SoDs allow one to specify how a user should be restricted in activating a role. For example, specifying the total duration for which a user is allowed to activate a role

SoD: separation of duty

access control decisions; i.e., they do not support the collaboration among various users to gain a specific permission, and cannot satisfy the practical requirements for collaborative access control.

3 Overview of the RBAC96 model

We will base our discussion on RBAC96 (Sandhu *et al.*, 1996), the most well-known role-based access control model. In this section, we provide an overview of the concepts within the model. Central to the model are the concepts of user, role, role hierarchy, permission, session, user-role assignment, permission-role assignment, and constraint (Fig. 1).

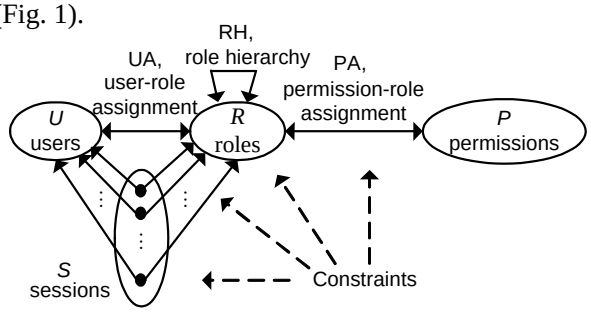

Fig. 1 The RBAC96 model

A user is a human being or an intelligent autonomous agent. A role is a job function or job title within an organization with some associated semantics regarding the authority and responsibility conferred on a member of the role. A role hierarchy is a partial order on roles. If role r_i can inherit permissions assigned to role r_j, denoted as $r_i \geq r_j$, we refer to r_i as a senior role of r_j, and r_j as a junior role of r_i. A permission is an approval of a particular mode of access to one or more objects in the system. Each session is a mapping of one user to possibly many roles. Users are associated with roles using the user-role assignment relation $UA \subseteq U \times R$. Permissions are associated with roles using the permission-role assignment relation $PA \subseteq P \times R$. Constraints are a powerful mechanism for laying out higher-level organizational policies.

4 Description of the CACAM model

In this section, we describe the central notions of our model (Fig. 2). CACAM preserves the advantages of RBAC, and offers the added specification of fine-grained collaborative access control policies by introducing the concepts of collaborative permission, collaboration constraint, and collaborative authorization policy. In CACAM, permissions are classified into two categories: regular permissions and collaborative permissions. Regular permissions refer to the permissions defined in traditional RBAC models, and can be granted without the participation of multiple independent parties. Collaborative permissions

refer to the permissions that can be accessed only when multiple-independent parties agree to do so. A collaborative permission is defined as a regular (RBAC) permission constrained by a collaboration constraint, which specifies the conditions that should be satisfied by collaborators for gaining it. Collaborative permissions are assigned to roles via the specification of collaborative authorization policies.

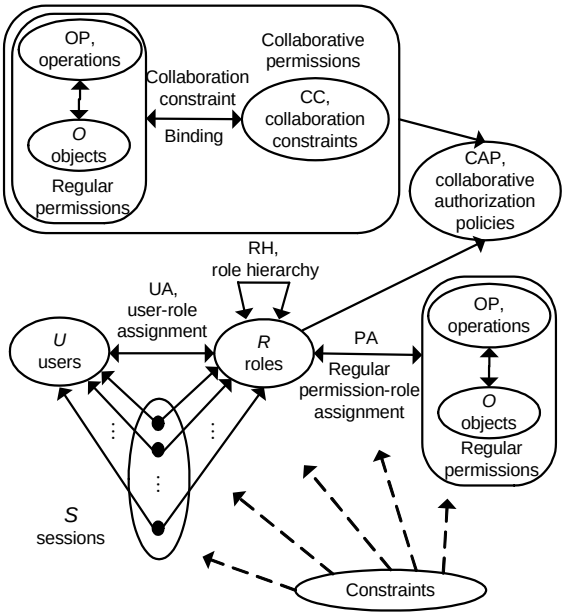

Fig. 2 The collaborative access control authorization model (CACAM)

4.1 Collaborative permission

Definition 1 (Regular permission) A regular permission p is defined as a pair p: <operation, object>, where operation is a particular mode of access, and object denotes a specific object.

Definition 2 (Collaborative permission) A collaborative permission can be defined as a regular permission associated with a collaboration constraint, which grants access only if the collaboration constraint evaluates to 'true'. A collaborative permission cp takes the form cp: <p, cc>, where p is a regular permission defined above, and cc denotes the collaboration constraint associated with cp (collaboration constraints will be discussed in more detail later).

For example, the permission to read confidential business contracts is a collaborative permission, and can be accessed only if at least three different users including a general manager agree to do so, and these users should activate at least two different roles, whereas the permission to read public documents is a regular permission.

4.2 Collaboration constraint

A collaborative permission to be accessed by multiple parties may have some specific requirements on the collaboration among these parties, which we refer to as collaboration constraints.

Definition 3 (Collaboration constraint) A collaboration constraint can be specified as a Boolean expression on some attributes of the collaborators involved in gaining corresponding collaborative permission, denoted as

$$\text{cc}:=\text{Clause}_1\vee\text{Clause}_2\vee\ldots\vee\text{Clause}_i,$$
$$\text{Clause}:=\text{Condition}_1\wedge\text{Condition}_2\wedge\ldots\wedge\text{Condition}_m,$$
$$\text{Condition}:=\text{<at><operator><value>},$$

where 'at' represents an attribute of collaborators in accessing a collaborative permission, 'operator' is a logical operator in the set {>, <, >=, <=, ==, !=} and this set can be extended to also accommodate user-defined operators, and 'value' is a specific value of 'at'.

These attributes can be the number of collaborators (col_num), the total weight of targeted permission owned by collaborators (total_weight), the number of roles activated by collaborators (role_num), and the set of roles activated by collaborators (role_set), etc. The possible attributes on collaborators can be defined flexibly according to the security requirements in collaborative access control.

For example, consider the following two collaborative permissions:

$$\text{cp}_1\text{: <read, confidential contract, cc}_1\text{>},$$
$$\text{cp}_2\text{: <read, regular contract, cc}_2\text{>},$$

where cc_1: (col_num>=3)$\wedge$(role_num>=2)$\wedge$(role_set $\supseteq$'general manager'), cc_2: col_num>=2.

cp_1 states that the permission to read confidential business contracts can be accessed only if at least three different users including a general manager agree to do so, and these users should activate at least two different roles; cp_2 indicates that the permission to read regular contracts can be granted to a requester only if he/she has approval from another user.

A collaboration constraint will be evaluated when there is a request to access the corresponding permission. Permission will be granted only if the collaboration among participants supporting the request satisfies the constraint. Different collaborative permissions may have different collaboration constraints. A collaborative permission with a higher degree of sensitivity (e.g., when the protected resource related to the permission is very sensitive), may require a more complex collaboration constraint. For instance, permission cp_1 shown above requires more collaborators to be involved than permission cp_2.

4.3 Context constraint

Constraints proposed in traditional RBAC models can be classified into three broad categories: SoD, cardinality constraints, and prerequisite constraints. These constraints cannot express contextual information (e.g., time and location) that can specify complicated situations in which roles can activate assigned permissions. For example, sales clerks can be granted the permission to read important business strategies with a weight of 1 only when access time is between 9 am and 5 pm. To address this issue, a context constraint is defined based on previous work on context-aware applications (Dey, 2001; Neumann and Strembeck, 2003; Hulsebosch *et al*., 2005).

Definition 4 (Context parameter) A context parameter represents a certain property of the environment, capable of influencing or controlling when and how a collaborative authorization policy is enforced. A context parameter may be a concrete property familiar in everyday life, such as time, location, identity, activity, and network capabilities, or an abstract concept such as the access intention of a user. By analyzing the system security requirements, security administrators can determine the set of context parameters used to specify authorization policies. For example, suppose we have a context parameter set CE={time, access_IP_address, DA_IPSet}, where time represents the current time, access_IP_address the IP address of a user, and DA_IPSet the IP address set of the local network.

Each context parameter is associated with a context function, which is a mechanism to acquire the current value of the parameter. For example, the function gettime() is defined to return the current time, and the function access_IPaddress(u) is used to obtain the IP address of user u, who may be the user initiating the current access request.

Definition 5 (Context condition) A context condition is a Boolean expression that compares the current value of a context parameter with either a predefined specific value, or the current value of another context parameter. It takes the form cd: <context_par> <operator><*v*>, where context_par is a context parameter with which the context condition is concerned, 'operator' is either a logical operator in the set {>, <, >=, <=, ==, !=}, or a user-defined operator as well, and v denotes a predefined value of context_par, or another context parameter. For example, 'cd_1: access_IP_address in DA_IPSet' is a context condition, indicating that the access IP address is in the local area network, where 'in' is a user-defined logical operator.

Definition 6 (Context constraint) A context constraint is defined as a logical conjunction of context conditions, and is denoted as

$$c:=\text{Clause}_1\vee\text{Clause}_2\vee\ldots\vee\text{Clause}_i,$$
$$\text{Clause}:=\text{cd}_1\wedge\text{cd}_2\wedge\ldots\wedge\text{cd}_n,$$

where cd_i (i=1, 2, …, n) represent context conditions defined above. For example, the following context constraint states that the current time is between 9 am and 5 pm, or that the access IP address is in the local area network:

c_1: ((time>=9:00)∧(time<=17:00))∨
(access_ IP_ address in DA_IPSet).

Before describing design details, the criteria for choosing representative nodes are investigated.

4.4 Collaborative authorization policy

In classical RBAC models, permissions are assigned to roles via the specification of permission-role assignment relations. If a permission is assigned to a role, then a member of the role can access the permission. This is rather different from the scenarios in collaborative access control. In collaborative access control, a collaborative permission is partitioned into several pieces and distributed to multiple users. In some cases, the permission can be accessed only if the number of collected permission pieces is equal to or greater than a required threshold-number. Moreover, the number of permission pieces owned by different users may be different since the degree of

trustworthiness in gaining the permission differs from one user to another. A higher-level user usually behaves with a higher degree of trust and thus is assigned more permission pieces. Consider the following scenario:

Scenario 1 Consider an enterprise with several roles, including general manager, sales manager, and sales clerk. To safeguard the confidentiality of business strategies, the permission to read such strategies is divided into several pieces for distribution to these roles. A general manager is assigned three pieces towards the permission, a sales manager has two pieces, and a sales clerk is assigned only one piece, since role 'general manager' behaves with the highest, and role 'sales clerk' the lowest degree of trust in collaborative access to the permission. An individual shall not be granted such a permission unless he/she has approval from one or more other users, and at least five pieces towards the permission are required.

In this scenario, role 'general manager' behaves with a higher degree of trustworthiness and is assigned more pieces towards the permission than other roles. Therefore, the permission can be accessed with fewer participants if a general manager is involved. Only two users, e.g., a general manager and a sales manager, are required to gain the permission with the participation of a general manager; otherwise, at least three users, e.g., two sales managers and a sales clerk, are required.

To address the above requirement in collaborative access control, we introduce the concept of 'collaborative authorization policy', which makes it possible to reflect the degree of trust of different roles in permission assignments.

Definition 7 (Collaborative authorization policy) A collaborative authorization policy, cap, is defined as a 5-ary tuple, i.e., cap: <*r*, cp, pweight, type, *c*>. Herein, cp is a collaborative permission and pweight represents the weight of permission cp owned by a member of role *r*, i.e., the number of permission pieces assigned to role *r* towards permission cp. pweight is identified based on the degree of trust of role *r* in accessing permission cp. 'type' denotes one of two policy types, either inheritable '1' or non-inheritable '0'. *c* is the context constraint related to cap, specifying the situation under which cap is active (i.e., a member of role *r* can be granted permission cp with a weight pweight if constraint *c* is satisfied). *c* is 'null' if there are no constraints for the activation of cap.

Definition 8 (Inheritable collaborative authorization policy) cap: <*r*, cp, pweight, 1, *c*> as an inheritable collaborative authorization policy states that role *r* and its senior roles can be granted permission cp with a weight pweight if constraint *c* is satisfied.

Definition 9 (Non-inheritable collaborative authorization policy) cap: <*r*, cp, pweight, 0, *c*> as a non-inheritable collaborative authorization policy means that role *r* can be granted permission *p* with a weight pweight if constraint *c* is satisfied, whereas its senior roles cannot inherit such a permission.

By the specification of inheritable attributes in collaborative authorization policies, CACAM can support full and partial authority inheritance on role hierarchies.

An example of collaborative authorization policy is as follows:

cap_1: <sales manager, cp_1, 2, 0, null>,
cp_1: <read, business strategy, cc_1>,
cc_1: (col_ num>=2)∧(total_weight>=5).

cap_1 is a non-inheritable policy, stating that role 'sales manager' is assigned the collaborative permission to read business strategies with a weight of 2, whereas its senior roles cannot inherit the permission.

4.5 Complexity of permission weight specification

The identification of permission weights is important for the definition of collaborative authorization policies. As described above, the weight of permission cp owned by a member of role *r* is identified based on the trust level of role *r* in accessing permission cp. A role with a higher degree of trustworthiness in gaining a specific collaborative permission should be assigned a larger weight over the permission. Therefore, permission weight specification is based on trust-level evaluation.

Trust is a subjective and elusive notion. As Gambetta (1990) pointed out: "trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent will perform a particular action." Trust is affected by those actions that we cannot monitor, and the level of trust depends on how our own actions are in turn affected by the agent's actions. Moreover, trust is a multi-dimensional quantity and is given for a specific con-

text. A role that is trusted to access a specific permission may not be trusted to access other permissions. For example, a designer who is trusted to design a drawing may not be trusted to approve a drawing. Thus, the same role may have different degrees of trustworthiness in accessing different permissions, and different roles may have different degrees of trust in accessing the same permission. Therefore, trust evaluation is a subjective and complex action, which depends on the security administrator's knowledge of the semantics of roles and collaborative permissions.

The degree of trustworthiness can be expressed as trust values. Different values have different semantics (Table 2).

Table 2 Trust values and their semantics

Trust value	Meaning
1	Lowest possible trust
2	Average trust level
3	High trust level, more trustworthy than most roles/users
4	Very high trust level

When the degree of trust of role r in accessing permission cp has been evaluated and represented as a trust value, the weight of permission cp owned by a member of role r can be specified as the trust value. Suppose that rn is the number of roles and cpn is the number of collaborative permissions in the system. The complexity of permission weight specification is O(rn×cpn).

4.6 Calculation of permission weight in the presence of role hierarchy

Let us consider the following policies:

cap_1: <general manager, cp_1, 2, 0, null>,
cap_2: <designer, cp_1, 1, 1, c_1>,
cp_1: <read, top secret drawing, cc_1>,
cc_1: (col_num>=2)∧(total_weight>=5)∧((role_set⊇'board chairman')∨(role_set⊇'general manager')),
c_1: (time>=9:00)∧(time<=17:00)∧(access_IP_address in DA_IPSet).

The permission to read top secret drawings is a collaborative permission, which cannot be granted unless at least two different users, including a board chairman or general manger, agree to do so. Moreover, the total weight of the permission owned by these collaborators should be equal to or greater than 5. Suppose that role 'general manager' is senior to role 'designer'. Policy cap_2 is inheritable; i.e., role 'general manager' can inherit the permission with a weight of 1 from its junior role 'designer'. At the same time, we observe that a general manager is directly assigned a weight of 2 towards the permission. In such a case, the calculation and identification of the total weight of the permission owned by the role 'general manager' is a significant issue. Here we are concerned mainly with the calculation of permission weight in the presence of role hierarchies.

Algorithm 1 Totalweight(r, cp)

Input: role r, collaborative permission cp
Output: the total weight of permission cp owned by role r

```
P←∅
tpw←0
/* if there exists a policy that is associated with role r and
 permission cp, and its context constraint is satisfied */
if there exists cap_i: <r_i, cp_i, pweight_i, type_i, c_i>∈CAP,
    r_i=r, cp_i=cp, and c_i is true then
  tpw←tpw+pweight_i
end if
/* Each policy in P is inheritable, its role is junior to r, its
 permission is cp, and its context constraint is satisfied */
P←{cap_j|cap_j: <r_j, cp_j, pweight_j, type_j, c_j>∈CAP, r_j≤r,
    cp_j=cp, type_j=1, and c_j is true}
for each cap_j: <r_j, cp_j, pweight_j, type_j, c_j>∈P do
  tpw←tpw+pweight_j
end for
return tpw
```

As described in Algorithm 1, in the presence of role hierarchies, the total weight of permission cp owned by a member of role r is the sum of the weight directly assigned to r and the weight inherited from its junior roles. The complexity of the algorithm is O(capn), where capn represents the number of defined collaborative authorization policies.

Theorem 1 Algorithm 1 is correct.

Proof Algorithm 1 evaluates all policies. If there is no active policy that is associated with permission cp and role r (or its junior roles), the algorithm will return 0. Otherwise, it returns the sum of the weight directly assigned to r and the weight inherited from its junior roles towards permission cp. The algorithm will terminate since the number of collaborative authorization policies is finite.

Therefore, in the above example, a general manager has a total weight of 3 towards collaborative

permission cp_1 when the current time is between 9 am and 5 pm, and the access IP address is in the local area network.

5 Conflict detection and resolution

For a large-scale system, the number of collaborative authorization policies may grow large and the interactions between them will be complex and difficult to predict. When two collaborative authorization policies are deemed incompatible with each other and their objectives cannot be simultaneously met, or the performance of two or more collaborative authorization policies will lead to situations that are forbidden by defined SoD constraints, conflicts will occur.

We believe that conflicts can be classified into two broad categories, defined as follows:

Definition 10 (Policy conflict) Policy conflicts occur when two or more collaborative authorization policies are deemed incompatible with each other. For example, consider the following two policies:

cap_1: <general manager, cp_1, 2, 0, null>,
cap_2: <general manager, cp_1, 1, 0, null>,
cp_1: <read, business strategy, cc_1>,
cc_1: (col_num>=2)∧(total_weight>=5).

Policy cap_1 states that a member of role 'general manager' is assigned the permission to read a business strategy with a weight of 2, while policy cap_2 indicates that a member of role 'general manager' is assigned the permission with a weight of 1. Thus, both of these policies cannot be satisfied at the same time and policy conflicts occur, and in this case, we cannot decide how much weight, 2 or 1, should be assigned to the role.

Definition 11 (Policy-constraint conflict) Policy-constraint conflicts occur when the specification of two or more collaborative authorization policies violates defined SoD constraints in the system. For example, consider the following two policies:

cap_1: <technique department manager, cp_1, 2, 0, null>,
cap_2: <technique department manager, cp_2, 1, 0, null>,
cp_1: <write, secret drawing, cc_1>, cc_1: (col_num >=2)∧(role_set⊇'general manager'),
cp_2: <approve, secret drawing, cc_2>, cc_2: (col_num>=3)∧(total_weight>=6)∧(role_set⊇'general manager').

Policy cap_1 states that role 'technique department manager' is assigned permission to write secret drawings with a weight of 2, while policy cap_2 indicates that role 'technique department manager' is assigned permission to approve secret drawings with a weight of 1. Thus, the performance of the two policies will introduce conflicts into the system since the same person must not be allowed both to write and approve the same drawing for the separation of duties.

The development of conflict detection and resolution methods that can ensure and maintain the consistency of all collaborative policies is a significant research challenge. In this section, we are concerned mainly with the provision of effective conflict detection and resolution rules for the above defined conflicts.

5.1 Conflict detection

5.1.1 Policy conflict detection

Definition 12 (Related context constraints) For two context constraints c_i and c_j, if the contextual information specified by c_i overlaps with the contextual information specified by c_j (e.g., an overlap between two time intervals and another overlap between two locations), then they are related context constraints, denoted as: Related(c_i, c_j).

Definition 13 (Non-related context constraints) For two context constraints c_i and c_j, if one of the following conditions is satisfied, then they are non-related context constraints, denoted as 'Non-related(c_i, c_j)':

1. $c_i \rightarrow \neg c_j$. The reverse of c_j is inferable by c_i; that is, wherever c_i is satisfied, c_j is not satisfied.
2. $c_j \rightarrow \neg c_i$. The reverse of c_i is inferable by c_j; that is, wherever c_j is satisfied, c_i is not satisfied.

For example, consider the following three constraints:

c_1: (time>=8:00)∧(time<=10:00),
c_2: (time>=9:00)∧(time <=10:00),
c_3: (time>=11:00)∧(time<=17:00).

We have Related(c_1, c_2), Non-related(c_1, c_3), Non-related(c_2, c_3).

A knowledge-based or a rule-based inference mechanism can be employed to perform the above inference.

Definition 14 (Related collaborative authorization policies) If two collaborative authorization policies cap_i: <r_i, cp_i, $pweight_i$, $type_i$, c_i> and cap_j: <r_j, cp_j, $pweight_j$, $type_j$, c_j> have an overlapping scope specified by the pre-condition $(r_i=r_j)\wedge(cp_i=cp_j)\wedge Related(c_i, c_j)$, then they are related collaborative authorization policies, denoted as 'Related(cap_i, cap_j)'.

Rule 1 (Policy conflict detection rule) $\forall cap_i, cap_j \in CAP$, $Related(cap_i, cap_j)\wedge((pweight_i\neq pweight_j)\vee(type_i \neq type_j))\rightarrow Conflict(cap_i, cap_i)$. That is, if two policies are related and have different weights or types, then they are inconsistent and policy conflicts occur. Policy conflicts are detected at specification time, when new policies are added.

Algorithm 2 PolicyConflictDetection(CAP, new_cap)

Input: the set of collaborative authorization policies CAP, newly defined policy new_cap: <*r*, cp, pweight, type, *c*>

Output: the set of policies exhibiting policy conflicts PolicyConflict_CAP

```
1  PolicyConflict_CAP←∅
2  for each cap_i: <r_i, cp_i, pweight_i, type_i, c_i>∈CAP do
3     if (r_i=r)∧(cp_i=cp)∧Related(c_i, c) then
4        if (pweight_i≠pweight)∨(type_i≠type) then
5           PolicyConflict_CAP←PolicyConflict_CAP∪
            (cap_i, new_cap)
6        end if
7     end if
8  end for
9  return PolicyConflict_CAP
```

The complexity of Algorithm 2 is $O(capn)$. Next, we will give a correctness proof of the algorithm.

Lemma 1 Two policies cap_i: <r_i, cp_i, $pweight_i$, $type_i$, c_i> and cap_j: <r_j, cp_j, $pweight_j$, $type_j$, c_j> will exhibit a conflict only if they are related.

Proof The policy cap_i states that role r_i has a weight of $pweight_i$ towards permission cp_i if constraint c_i is satisfied. The policy cap_j states that role r_j has a weight of $pweight_j$ towards permission cp_j if constraint c_j is satisfied. If the two policies are not related, then one of the following conditions holds: (1) $r_i\neq r_j$; (2) $cp_i\neq cp_j$; (3) Non-related(c_i, c_j).

Note that if the two policies are not related, then they have different permissions, roles, or non-related context constraints. Clearly they would exhibit no conflict in such cases. As Moffett and Sloman (1994) pointed out, overlap is crucial to the discussion of policy conflicts. Without a certain overlap between the objects in two policies, there can be no conflict between them.

If the two policies are related, then they have a triple overlap of permission, role, and context constraint, and in such case, there is a potential for policy conflicts. For example, if they have different weights, i.e., $pweight_i\neq pweight_j$, then we cannot decide how much weight, $pweight_i$ or $pweight_j$, the role r_i (r_j) has towards permission cp_i (cp_j) under the situation in which c_i and c_j are simultaneously met, and a policy conflict occurs. Therefore, we can conclude that only if two policies are related will they exhibit a policy conflict.

Lemma 2 Two related policies cap_i and cap_j will exhibit a policy conflict only if the condition $(pweight_i\neq pweight_j)\vee(type_i\neq type_j)$ holds.

Proof If two related policies cap_i and cap_j have different weights, then we cannot decide how much weight, $pweight_i$ or $pweight_j$, the role r_i has towards permission cp_i under the situation in which c_i and c_j are simultaneously met, and a policy conflict occurs. If they are of a different type, i.e., one is inheritable and the other is non-inheritable, then we cannot decide whether the senior roles of r_i can inherit the permission cp_i under the situation in which c_i and c_j are simultaneously met, and a policy conflict occurs. If two related policies do not satisfy the condition defined above, then they define the same weight for the combination of role r_i and permission cp_i, and have the same type. In this case, they would exhibit no conflict.

Theorem 2 Algorithm 2 is correct.

Proof We can conclude form Lemmas 1 and 2 that only if two collaborative authorization policies are related and have different weights or types, will they exhibit policy conflicts. Moreover, the algorithm will terminate since the number of collaborative authorization policies is finite.

5.1.2 Policy-constraint conflict detection

Policy conflicts can be detected purely by syntactic analysis of the policies. However, the detection of policy-constraint conflicts depends on the understanding of the semantics of policies and cannot be

determined directly from policy specifications. Owing to the complexity and diversity of SoD constraints, detecting policy-constraint conflicts is a complex topic, and in this subsection, we focus on providing conflict detection methods for conflicts exhibited by collaborative authorization policies and SSoD (static SoD) contraints. We will address other types of constraints (e.g., cardinality constraints and dynamic SoD variations) in future work.

SSoD is also called strong exclusion, indicating that the same user can be assigned to no more than one role in a mutually exclusive role set, or the same role/user cannot be assigned two or more permissions in a mutually exclusive permission set. SSoD constraints concern rules about how the users and permissions assigned to different roles may overlap and are checked at specification time to prevent the construction of any authorization policy that may violate the constraints. To detect policies that are in conflict with SSoD constraints, we define the concept of the policy-specification rule.

Definition 15 (Policy-specification rule) Policy-specification rules are used to detect the conflicts arising from the specification of collaborative authorization policies that violate SSoD constraints. Each policy-specification rule is associated with a SSoD constraint and takes the form: psr(sc): $\forall cap_i$, $cap_j \in CAP$, $C \rightarrow Conflict(cap_i, cap_i)$, where sc denotes the SSoD constraint. The rule states that if two policies cap_i and cap_j satisfy the conditions specified in expression C, then they exhibit a policy-constraint conflict.

For example, consider a SSoD constraint sc_1 stating that users are not allowed both to draft and to approve contracts. The constraint can also be stated as: "Two policies that have overlapping roles, and grant permissions to draft and approve contracts respectively, should not exist." This can be defined as a policy-specification rule shown below:

$psr(sc_1)$: $\forall cap_i, cap_i \in CAP, (Role(cap_i) \cap Role(cap_j) \neq \varnothing) \wedge (cp_i = <draft, contract>) \wedge (cp_j = <approve, contract>) \rightarrow Conflict(cap_i, cap_i)$,

where $Role(cap_i)$ denotes the set of roles related to policy cap_i. If cap_i is non-inheritable, then $Role(cap_i) = r_i$. If it is inheritable, then $Role(cap_i) = r_i \cup \{r_j | r_j \geq r_i\}$.

Policy-specification rules are evaluated at specification time, when new policies are added.

Algorithm 3 PolicyConstraintConflictDetection(CAP, PSR, new_cap)

Input: CAP, the set of policy-specification rules PSR, and newly added policy new_cap
Output: the set of policies exhibiting policy-constraint conflicts PC_Conflict

```
1 PC_Conflict←∅
2 for each cap_i∈CAP do
3   for each psr∈PSR do
4     if cap_i and new_cap satisfy the condition expression
        defined in psr then
5       PC_Conflict←PC_Conflict∪(cap_i, new_cap)
6     end if
7   end for
8 end for
9 return PC_Conflict
```

Suppose that the number of policy-specification rules is psrn. The complexity of the algorithm is $O(capn \times psrn)$.

Theorem 3 Algorithm 3 is correct.

Proof According to Definition 15, if two policies satisfy the condition specified in a policy-specification rule, then they violate the SoD constraint related to the rule, and a policy-constraint conflict occurs. Algorithm 3 compares every policy pair composed of an existing policy and the new policy with policy-specification rules. If these rules are correctly defined, then the algorithm will return all policies exhibiting policy-constraint conflicts. The algorithm will terminate since CAP and PSR are both finite sets.

5.2 Conflict resolution

If conflicts occur, it becomes necessary to apply methods for conflict resolution. A practical method for resolving conflicts is to identify which policy involved in a conflict situation will take precedence. Some methods that can be used to establish precedence in policy conflict situations have been studied in policy-based management systems including negative policy precedence and assigning explicit priorities to policies (Dunlop *et al*., 2003; He *et al*., 2005), but they can produce valid results in only some circumstances and are not suitable for our collaborative authorization policies.

To address this issue, based on our previous research (Ma *et al*., 2009), the concepts of 'priority rule' and 'resolution policy' are introduced in this work, allowing security administrators to decide the methods for conflict resolution flexibly according to the system requirements.

Definition 16 (Priority rule) A priority rule defines a method to identify which policy involved in a conflict situation will take precedence. It can be formally described as priority_rule: <cap_i, relation, cap_j>→ priority(cap_i)>priority(cap_j), where cap_i and cap_j are two policies involved in a conflict situation, and 'relation' denotes a possible relationship between them. For example, <cap_i, $>_{timestamp}$, cap_j> represents that the creation date of cap_i is later than that of cap_j, and <cap_i, $<_{pweight}$, cap_j> indicates that the permission weight of cap_i is smaller than that of cap_j. The possible relationships between two policies can be defined flexibly according to the conflict resolution requirements of the system. The rule states that if two policies cap_i and cap_j satisfy the relationship specified by 'relation', then cap_i precedes cap_j.

Definition 17 (Resolution policy) A resolution policy is a total order of some priority rules that can be formally described as resolution_policy: (priority_$rule_1$>priority_$rule_2$>…>priority_$rule_n$). Therein priority_$rule_t$ (t=1, 2, …, n) are the priority rules integrated in the resolution policy, and '>' denotes the order of priority established for the rules. Conflict resolution is a step-by-step process. Sequentially, a priority rule of higher priority is selected to resolve remaining conflicts at each step, until no conflict remains.

For example, there is a conflict resolution policy defined as resolution_policy: (priority_$rule_1$>priority_$rule_2$>priority_$rule_3$).

priority_$rule_1$: <cap_i, $>_{timestamp}$, cap_j>→
priority(cap_i)>priority(cap_j),
priority_$rule_2$: <cap_i, $>_{granter\_level}$, cap_j>→
priority(cap_i)>priority(cap_j),
priority_$rule_3$: <cap_i, $<_{pweight}$, cap_j>→
priority(cap_i)>priority(cap_j).

The resolution policy represents that when two policies cap_i and cap_j exhibit a conflict, first we will compare the timestamps of the two conflict policies, and a new policy will override an old policy. If they have the same timestamp, then we compare the level of the two granters, and a higher-level granter overrides a lower-level granter. If the level of the two granters cannot be compared, then the policy with smaller weight will take precedence.

The maximum number of conflicts exhibited by all collaborative authorization policies is 1/2×capn×(capn−1). The complexity of resolving all these conflicts using the proposed conflict resolution method is $O(m\times capn^2)$, where m is the number of priority rules in the resolution policy.

6 Implementation framework of CACAM

Fig. 3 gives a high-level overview of the implementation framework of CACAM.

The framework consists of a certificate authority (CA) and a collaborative authorization server (CAS). The CA aims to create an X.509 access certificate for each legitimate user. The CAS is responsible for deciding whether an access request will be granted. When a user wants to access a collaborative permission, the user sends his/her access request and the access certificate issued by CA to CAS, and CAS makes an access decision, i.e., to grant or deny the request. First, CAS verifies the validity of the access certificate, and authenticates the requester. Then, it evaluates defined collaborative authorization policies applicable to the request. If the requester cannot satisfy the collaboration constraint of the targeted permission, then it checks all signed trust delegation

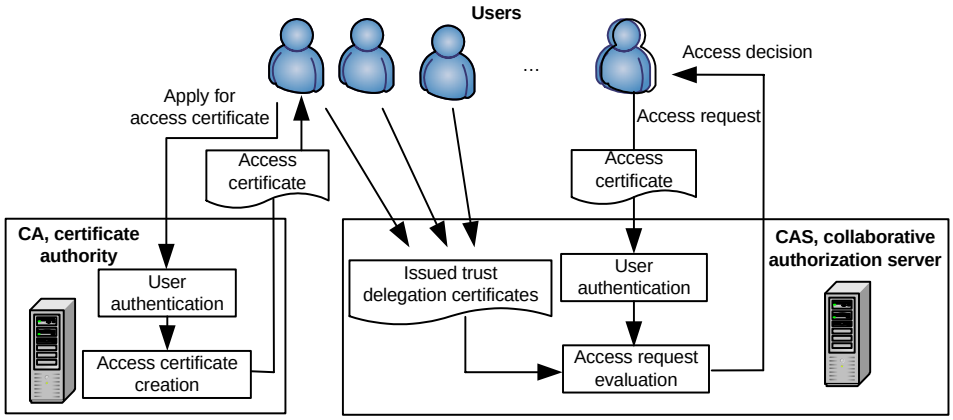

Fig. 3 The implementation framework of the collaborative access control authorization model (CACAM)

certificates to identify collaborators who are willing to support the request. A user can issue a trust delegation certificate if he/she trusts another user to gain specific collaborative permission. The certificate is signed with the issuer's private key. All signed trust delegation certificates are stored in CAS for future access request evaluation. Finally, CAS decides whether the collaborators permitting the request satisfy the collaboration constraint of the requested permission or not, and makes an access decision.

6.1 Access certificate

Definition 18 (Role identity code) In CACAM, each role has a unique identity code represented by a prime number, e.g., 3, 5, and 7. The identity codes of two roles must be different. In the following sections, let ic_i represent the identity code of role r_i.

Definition 19 (Role assignment parameter) A role assignment parameter is used to express all roles assigned to a user. The role assignment parameter of user u is the product of the identity codes of all roles assigned to u, i.e., $ra = \prod_{r_i \in P} ic_i$, where P denotes the set of roles assigned to u.

Definition 20 (Access certificate) A user must apply for an access certificate issued by CA before he/she accesses permissions. Access certificates are used to authenticate securely a user that is requesting access to a permission. Fig. 4 shows the contents of an access certificate. Fig. 5 shows the process of issuing access certificates.

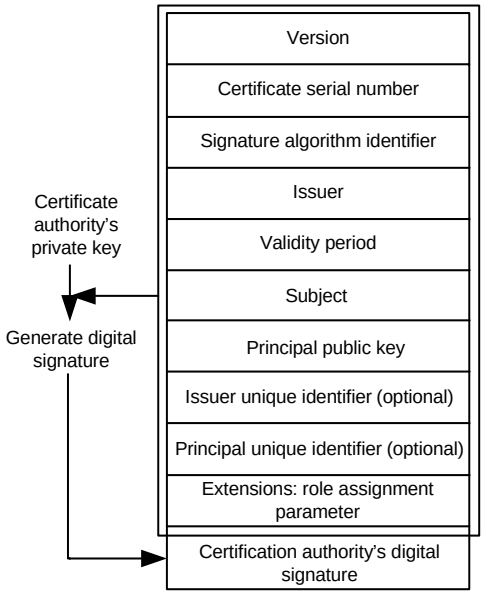

Fig. 4 X.509 access certificate

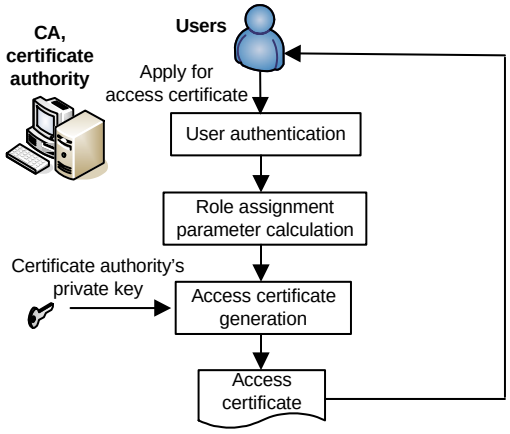

Fig. 5 The process of issuing access certificates

Furthermore, since a change to the system state (e.g., the deletion of an existing role) may cause an access certificate to be invalid, a table of invalid access certificates is created and stored in CAS to present a user from accessing permissions using an invalid certificate. For example, if a user has been revoked from a role following a change in organizational circumstances, his/her access certificate will be invalid.

6.2 Access request evaluation

Fig. 6 shows the authorization process in CACAM.

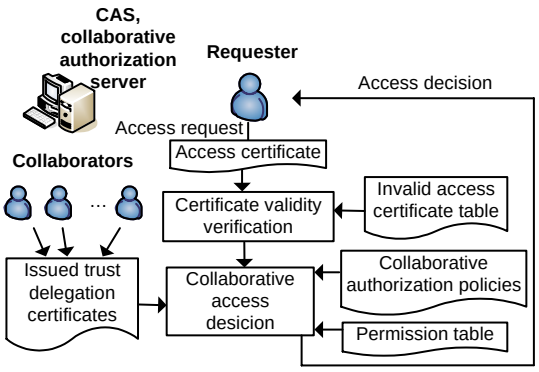

Fig. 6 The process of data access in the collaborative access control authorization model (CACAM)

When a user requests to be granted a collaborative permission, the user sends his/her access request and access certificate to CAS. First, CAS verifies the validity of the certificate. If the certificate is invalid, then the request will be denied; otherwise, CAS evaluates all collaborative authorization policies and calculates the total weight of the targeted permission owned by the role activated by the requester. If the

requester can satisfy the collaboration constraint of the targeted permission, then CAS grants the access request; otherwise, CAS evaluates all policies, identifies the set of roles and users that have a weight over the targeted permission, and checks the list of signed trust delegation certificates. If there exists a user who has been assigned a weight over the requested permission and he/she has issued a trust delegation certificate associated with the requester and the targeted permission (i.e., he/she trusts the requester to access the targeted permission), then CAS will verify the validity of the trust delegation certificate. If the certificate is valid, then the user is a collaborator supporting the request. If all users permitting the request satisfy the collaboration constraint of the targeted permission, then CAS grants the request; otherwise, the request will be denied.

6.2.1 Trust delegation certificate

Definition 21 (Trust delegation certificate) A trust delegation certificate is issued when a user trusts a requester to gain specific collaborative permission. A trust delegation certificate takes the form 'tdc: $ER_{KR_{issuer}}$[Issuer, Role, subject, cp, Trust_Value, $ValidDate_s$, $ValidDate_e$]', where Issuer is the user who issues the certificate, Role denotes the role to which the issuer is assigned, 'subject' is a specific user who is the target of the certificate (i.e., the issuer is willing to support subject in gaining the targeted collaborative permission), cp is the targeted collaborative permission of the certificate, Trust_Value is used to represent the level of trust the issuer may have in 'subject' in accessing cp (i.e., how much 'subject' is trusted by the issuer to access cp), $ValidDate_s$ represents the starting date on which the certificate can be considered valid, $ValidDate_e$ represents the ending date on which the certificate will be considered invalid, ER represents the digital signature algorithm, and KR_{issuer} represents the private key of the issuer.

Trust delegation certificates are issued based on defined collaborative authorization policies. Only if there is a collaborative authorization policy that is associated with the combination of role r and collaborative permission cp, could a user who is a member of role r issue a trust delegation certificate over cp. A list of all issued trust delegation certificates is stored in CAS for access request evaluation.

Trust values and their meanings used in our approach are given in Table 3. The same issuer may have different levels of trustworthiness in the same requester to access different collaborative permissions. For example, a requester who is trusted to design a drawing may not be trusted to approve a drawing.

Table 3 Trust values and corresponding descriptions

Value	Meaning	Description
1	Minimal	Lowest possible trust
2	Average	Most users have this trust level
3	Good	More trustworthy than most users
4	Complete	Completely trust this user

Examples of trust delegation certificates are as follows:

tdc_1: $ER_{KR_{u2}}$[u_2, general manager, u_8, <write, secret drawing>, 3, 2007/08/31, 2009/09/01];

tdc_2: $ER_{KR_{u2}}$[u_2, general manager, u_{10}, <write, secret drawing>, 1, 2007/08/31, 2008/02/30].

The first certificate states that user u_2 who is a member of role 'general manager' has a trust level of 'Good' in user u_8 to write secret drawings, whereas the second certificate means that user u_2 has a trust level of 'Minimal' in user u_{10} to write secret drawings.

Definition 22 (Trustworthiness threshold) A trustworthiness threshold is used to decide whether the issuer of a trust delegation certificate could be a collaborator supporting the target user to gain the targeted permission of the certificate. When a requester requires access to a collaborative permission, CAS will check all signed trust delegation certificates to identify collaborators agreeing with the request. Suppose that there exists a certificate that is associated with the requester and the requested permission. Only if the trust-value specified in the certificate is equal to or greater than the predefined threshold-number, could the issuer of the certificate be a collaborator supporting the request. Security administrators can set a threshold-number for trustworthiness according to system security requirements.

Let us assume a trustworthiness threshold of Trust_Thres=2. According to the above certificates, if user u_{10} requests to be granted the collaborative permission to write a secret drawing, user u_2 will not be deemed as a collaborator of the request since the trust-value specified in certificate tdc_2 is less than Trust_Thres.

6.2.2 Access request evaluation algorithm

Algorithm 4 is used to determine whether a particular user's request to access a collaborative permission is allowed. In the following sections, let ER, DR, KR_u, and KU_u represent the RSA encryption algorithm, RSA decryption algorithm, private key of user u, and public key of user u, respectively.

Algorithm 4 AccessRequest_Evaluation(rm, ac)

Input: access request rm, and the requester's access certificate ac. rm takes the form 'rm: $ER_{KR_u}[u, r, cp, N_u]$', where u is the requester, r is the role activated by requester u, cp is the targeted collaborative permission, and N_u is a random number

Output: If the access request is granted, then return true; otherwise, return false

```
1  tw←0 /* tw denotes the total weight of permission cp
        owned by all collaborators supporting the request */
2  PR←∅ /* PR represents the set of roles that have a weight
        over the requested permission cp */
3  PU←∅ /* PU represents the set of users that have a weight
        over the requested permission cp */
4  Col←∅ /* Col represents the set of users supporting the
        request */
5  if ac is an invalid certificate then /* verify ac's validity */
6     return false
7  end if
   /* check whether the requester u is a member of role r */
8  if the role assignment parameter of ac is not divisible by the
      identity code of role r then
9     return false
10 end if
11 DR_KU_u [rm] /* decrypt rm with the public key of u */
12 check random number N_u and decide whether the request is
      a replay attack
13 if the request is a replay attack then
14    return false
15 end if
/* calculate the total weight of permission cp owned by role r */
16 tw←tw+Totalweight(r, cp)
/* if role r cannot be granted the targeted permission cp */
17 if tw==0 then
18    return false
19 end if
20 if requester u satisfies the collaboration constraint of per-
      mission cp then
21    return true
22 end if
   /* identify the set of roles and users that have a weight over
      the requested permission cp */
23 PR←∪{r_j|r_j∈Role(cap_i), cap_i: <r_i, cp_i, pweight_i, type_i,
      c_i>∈CAP, cp_i=cp}
24 PU←∪{u_j|u_j∈UA(r_j), r_j∈PR, u_j≠u} /* UA(r_j) represents
      the set of users assigned role r_j */
25 for each user u_j∈PU do
26    evaluate all issued trust delegation certificates and decide
      whether there exists a certificate that is issued by u_j and
      associated with requester u and targeted permission cp
27    if there exists such a certificate as 'tdc: ER_KR_ij[u_j, r_j, u,
      cp, Trust_Value, ValidDate_s, ValidDate_e]' (i.e., u_j is
      willing to support requester u to gain cp) and the cer-
      tificate is valid then
      /* check whether the trust-value reflecting the trustworthi-
      ness u_j has in requester u in accessing targeted permission
      cp is equal to or greater than the defined trustworthiness
      threshold */
28       if Trust_Value≥Trust_Thres then
29          Col←Col∪{<u_j, r_j>} /* add <u_j, r_j> to the set of
                collaborators permitting the request */
30          tw←tw+Totalweight(r_j, cp)
31       end if
32    end if
33 end for
34 if all users permitting the request satisfy the collaboration
      constraint of permission cp then
35    return true
36 else
37    return false
38 end if
```

The complexity of the algorithm is $O(capn \times rn^2 \times un)+O(I)$, where un is the number of users, rn is the number of roles, and $O(I)$ is the complexity of the RSA algorithm.

Theorem 4 Algorithm 4 is correct.

Proof An access request can be granted only if the following three conditions hold:

1. The requester is a legitimate user of the system.

2. The role activated by the requester has been assigned the requested permission via the specification of collaborative authorization policies; i.e., the total weight of the permission owned by the role should not be equal to 0.

3. The collaborators supporting the request satisfy the collaboration constraint of the requested permission.

The algorithm grants an access request only if the above conditions hold true.

Moreover, the algorithm will terminate. If the requester is illegitimate or his/her access certificate is invalid, the algorithm will terminate. Otherwise, the algorithm evaluates defined collaborative authorization policies, checks all signed trust delegation certificates, and decides whether the request will be granted or denied. Since the number of collaborative authorization policies and the set of users are finite, the algorithm will stop.

7 An illustrative example

An illustrative example we will use is an electrical design document management system. Fig. 7 shows the roles and the role hierarchy in the system.

Electrical design documents in the system are classified into three security levels, 'top secret', 'secret', and 'public', where 'top secret' is the highest security level and 'public' is the lowest. Permissions to access public documents are regular permissions (i.e., public documents can be accessed by a requester without the participation or agreement of other users), whereas permissions for accessing documents with level 'top secret' or 'secret' are collaborative permissions (i.e., these documents need to be accessed collaboratively by multiple users). Table 4 shows collaborative permissions for accessing top secret documents and corresponding collaboration constraints.

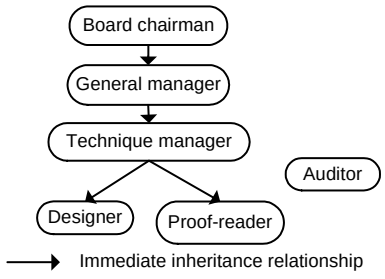

Fig. 7 Example roles and role hierarchy for an electrical design document management system

Related collaborative authorization policies in the system are defined in Table 5.

First, we will give an example of access certificate creation. Table 6 shows the user-role assignments in the system.

Assume that user u_3 sends a request to CA to apply for an access certificate. If u_3 is already authenticated, then CA will calculate the role assignment parameter for u_3, ra(u_3)=3×5=15, where 3 and 5 are the identity codes of designer and auditor, respectively. Finally, CA will create an X.509 access certificate for u_3.

Next, we show an example of access request evaluation. Assume that u_3 activates the role 'designer' and wants to read drawing *d* with level 'top secret'. First, u_3 sends the request and his/her access certificate to CAS. If the certificate is valid, then CAS

Table 4 Example collaborative permissions and corresponding collaboration constraints

Collaborative permission	Corresponding collaboration constraint	Description
cp_1: <read, top secret document, cc_1>	cc_1: (col_num>=2)∧(total_weight>=5)∧((role_set⊇'board chairman')∨(role_set⊇'general manager'))	The permission to read top secret documents cannot be granted to a requester unless the requester has approval from other users including a board chairman or general manager, and the total weight of the permissions owned by them is equal to or greater than 5
cp_2: <print, top secret document, cc_2>	cc_2: (col_num>=3)∧(role_num>=2)∧(total_weight>=6)∧(role_set⊇'board chairman')	The permission to print top secret documents cannot be granted to a requester unless the requester has approval from at least two other users. Moreover, the requester and the participants should activate at least two different roles including the role 'board chairman', and the total weight of the permissions owned by them is equal to or greater than 6
cp_3: <write, top secret document, cc_3>	cc_3: (col_num>=3)∧(role_num>=2)∧(total_weight>=5)∧((role_set⊇'board chairman')∨(role_set⊇'general manager'))	The permission to write top secret documents cannot be granted to a requester unless the requester has approval from at least two other users. Moreover, the requester and the participants should activate at least two different roles including the role 'board chairman' or 'general manager', and the total weight of the permissions owned by them is equal to or greater than 5
cp_4: <proof-read, top secret document, cc_4>	cc_4: (col_num>=3)∧(total_weight>=5)∧((role_set⊇'board chairman')∨(role_set⊇'general manager'))	The permission to proof-read top secret documents cannot be granted to a requester unless the requester has approval from at least two other users including a board chairman or general manger, and the total weight of the permissions owned by them is equal to or greater than 5
cp_5: <approve, top secret document, cc_5>	cc_5: (col_num>=3)∧(total_weight>=5)∧((role_set⊇{board chairman, technique manager})∨(role_set⊇{general manager, technique manager}))	The permission to approve top secret documents cannot be granted to a requester unless the requester has approval from at least two other users. Moreover, the collaborators supporting the request should include a board chairman and a technique manger or a general manger and a technique manger, and the total weight of the permissions owned by them is equal to or greater than 5

Table 5 Examples of collaborative authorization policies

Collaborative authorization policy	Description
cap_1: <board chairman, cp_1, 3, 0, null>	A board chairman is assigned a weight of 3 towards collaborative permission cp_1
cap_2: <general manager, cp_1, 2, 0, null>	A general manager is assigned a weight of 2 towards collaborative permission cp_1
cap_3: <technique manager, cp_1, 1, 0, cc_1>, cc_1: (time>=9:00)∧(time<=17:00)	A technique manager is assigned a weight of 1 towards collaborative permission cp_1 when access time is between 9 am and 5 pm
cap_4: <designer, cp_1, 1, 1, cc_2> cap_5: <proof-reader, cp_1, 1, 0, cc_2> cap_6: <auditor, cp_1, 1, 0, cc_2> cc_2: (time>=9:00)∧(time<=17:00)∧(access_IP_address in DA_IPSet)	A role designer, proof-reader, or auditor is assigned a weight of 1 towards cp_1 when access time is between 9 am and 5 pm, and the access IP address belongs to the local area network. Since cap_4 is inheritable, the senior roles of designer, e.g., technique manager, can inherit a weight of 1 towards cp_1 when the access time is between 9 am and 5 pm, and the access IP address belongs to the local area network
cap_7: <board chairman, cp_2, 3, 0, null>	A board chairman is assigned a weight of 3 towards collaborative permission cp_2
cap_8: <general manager, cp_2, 2, 0, null>	A general manager is assigned a weight of 2 towards collaborative permission cp_2
cap_9: <technique manager, cp_2, 1, 0, null>	A technique manager is assigned a weight of 1 towards collaborative permission cp_2
cap_{10}: <board chairman, cp_3, 2, 0, null>	A board chairman is assigned a weight of 2 towards collaborative permission cp_3
cap_{11}: <general manager, cp_3, 2, 0, null>	A general manager is assigned a weight of 2 towards collaborative permission cp_3
cap_{12}: <technique manager, cp_3, 1, 0, cc_1>	A technical manager is assigned a weight of 1 towards cp_3 when the access time is between 9 am and 5 pm
cap_{13}: <designer, cp_3, 1, 1, cc_2>	A designer and its senior roles can be granted a weight of 1 towards cp_3 when the access time is between 9 am and 5 pm, and the access IP address belongs to the local area network
cap_{14}: <board chairman, cp_4, 2, 0, null>	A board chairman is assigned a weight of 2 towards collaborative permission cp_4
cap_{15}: <general manager, cp_4, 2, 0, null>	A general manager is assigned a weight of 2 towards cp_4
cap_{16}: <technique manager, cp_4, 1, 0, cc_1>	A technique manager can be granted a weight of 1 towards cp_4 when the access time is between 9 am and 5 pm
cap_{17}: <proof-reader, cp_4, 1, 1, cc_2>	A proof-reader and its senior roles can be granted a weight of 1 towards cp_4 when the access time is between 9 am and 5 pm, and the access IP address belongs to the local area network
cap_{18}: <board chairman, cp_5, 2, 0, null>	A board chairman is assigned a weight of 2 towards collaborative permission cp_5
cap_{19}: <general manager, cp_5, 2, 0, null>	A general manager is assigned a weight of 2 towards cp_5
cap_{20}: <technique manager, cp_5, 1, 0, cc_1>	A technique manager can be granted a weight of 1 towards cp_5 when the access time is between 9 am and 5 pm
cap_{21}: <auditor, cp_5, 1, 0, cc_2>	An auditor can be granted a weight of 1 towards cp_5 when access time is between 9 am and 5 pm, and the access IP address belongs to the local area network

Table 6 User-role assignments in the example system

User	Explicitly assigned role(s)
u_1	Board chairman
u_2	General manager
u_3	Designer, auditor
u_4	Technique manager
u_5	Designer
u_6	Proof-reader
u_7	Auditor

will evaluate all policies and decide whether u_3 can satisfy the collaboration constraint of requested permission cp_1. Since permission cp_1 can be granted only when at least two different users, including a board chairman or general manager, agree to do so, CAS will check the list of issued trust delegation certificates and collect those that support the access request. Let us assume that there exist the following two trust delegation certificates associated with the request; i.e., the target user of these certificates is the requester, and their targeted permission is the requested permission:

tdc_1: $ER_{KR_{u1}}$[u_1, board chairman, u_3, <read, top secret drawing>, 2, 2008/03/01, 2009/09/01];

tdc_2: $ER_{KR_{u2}}$[u_2, general manager, u_3, <read, top secret drawing>, 2, 2008/08/31, 2009/12/30].

Let us assume a trustworthiness threshold of Trust_Thres=2. Both u_1 and u_2 have a trust-value of 2 in u_3 to gain the requested permission; i.e., they permit the request. Since u_1 is assigned a weight of 3, and u_2 a weight of 2 towards permission cp_1, all users

permitting the request satisfy the collaboration constraint of permission cp_1 and CAS will grant the request.

8 Conclusions and future work

In this paper, we have proposed an authorization model for collaborative access control (CACAM). In CACAM, permissions are assigned to roles via the specification of collaborative authorization policies, in which a permission weight reflecting the degree of trust in a role in gaining a collaborative permission is considered and a context constraint used to guarantee the principle of least privileges is defined. Each collaborative permission is related to a collaboration constraint that must be satisfied by requesters to gain the permission, including the least number of collaborators and the least total weight owned by collaborators. CACAM can satisfy the special requirements of collaborative access control. The implementation framework of CACAM is also presented. Effective conflict detection and resolution methods are provided to guarantee the exemption of inconsistency and ambiguities within collaborative authorization policies.

Our future work will address several issues including the administration of the CACAM model and the further refinement of collaborative authorization policies, which are considered to exist at many different levels of abstraction.

References

Ahn, G.J., Sandhu, R., 2000. The RCL 2000 Language for Specifying Role-Based Authorization Constraints. PhD Thesis, George Mason University, Fairfax, Virginia, USA.

Alsulaiman, F.A., Miege, A., EI Saddik, A., 2007. Threshold-Based Collaborative Access Control. Proc. Int. Symp. on Collaborative Technologies and Systems, p.45-56. [doi:10.1109/CTS.2007.4621737]

Ardagna, C.A., Cremonini, M., de Capitani di Vimercati, S., Samarati, P., 2008. A privacy-aware access control system. *J. Comput. Secur.*, **16**(4):369-397. [doi:10.3233/JCS-2008-0328]

Carminati, B., Ferrari, E., 2008. Privacy-Aware Collaborative Access Control in Web-Based Social Networks. Proc. 22nd Annual IFIP WG 11.3 Working Conf. on Data and Applications Security, p.81-96. [doi:10.1007/978-3-540-70567-3_7]

Crampton, J., 2003. Specifying and Enforcing Constraints in Role-Based Access Control. Proc. 8th ACM Symp. on Access Control Models and Technologies, p.43-50. [doi:10.1145/775412.775419]

Dey, A.K., 2001. Providing Architectural Support for Building Context-Aware Applications. PhD Thesis, Georgia Institute of Technology, Atlanta, Georgia, USA.

Dunlop, N., Indulska, J., Raymond, K., 2003. Methods for Conflict Resolution in Policy-Based Management System. Proc. 7th Int. Enterprise Distributed Object Computing Conf., p.98-109. [doi:10.1109/EDOC.2003.1233841]

Franz, E., Wahrig, H., Boettcher, A., Borcea-Pfitzmann, K., 2006. Access Control in a Privacy-Aware eLearning Environment. Proc. 1st Int. Conf. on Availability, Reliability and Security, p.879-886. [doi:10.1109/ARES.2006.20]

Gambetta, D., 1990. Can We Trust Trust? *In*: Gambetta, D. (Ed.), Trust: Making and Breaking Cooperative Relations. Basil Blackwell, Oxford, p.213-237.

Gligor, V.D., Gavrila, S., Ferraiolo, D., 1998. On the Formal Definition of Separation of Duty Policies and Their Composition. Proc. IEEE Computer Society Symp. on Research in Security and Privacy, p.172-183. [doi:10.1109/SECPRI.1998.674833]

He, Z.L., Tian, J.D., Zhang, Y.S., 2005. Analysis, detection and resolution of policy conflict. *J. Lanzhou Univ. Technol.*, **31**(5):83-86 (in Chinese).

Hulsebosch, R.J., Salden, A.H., Bargh, M.S., Ebben, P.W.G., Reitsma, J., 2005. Context Sensitive Access Control. Proc. 10th ACM Symp. on Access Control Models and Technologies, p.111-119. [doi:10.1145/1063979.1064000]

Joshi, J.B.D., Bertino, E., Shafiq, B., Ghafoor, A., 2003. Dependencies and Separation of Duty Constraints in GTRBAC. Proc. 8th ACM Symp. on Access Control Models and Technologies, p.51-64. [doi:10.1145/775412.775420]

Kim, K.I., Ko, H.J., Choi, W.G., Lee, E.J., Kim, U.M., 2008. A Collaborative Access Control Based on XACML in Pervasive Environments. Proc. Int. Conf. on Convergence and Hybrid Information Technology, p.7-13. [doi:10.1109/ICHIT.2008.225]

Koch, M., Mancini, L.V., Parisi-Presicce, F., 2002. A graph based formalism for RBAC. *ACM Trans. Inf. Syst. Secur.*, **5**(3):332-365. [doi:10.1145/545186.545191]

Li, D., Rao, P., Bertino, E., Li, N.H., Lobo, J., 2008. Policy Decomposition for Collaborative Access Control. Proc. 13th ACM Symp. on Access Control Models and Technologies, p.103-112. [doi:10.1145/1377836.1377853]

Li, E.Y., Du, T.C., Wong, J.W., 2007. Access control in collaborative commerce. *Decis. Support Syst.*, **43**(2):675-685. [doi:10.1016/j.dss.2005.05.022]

Ma, C.H., Lu, G.D., Qiu, J., 2009. Conflict detection and resolution for authorization policies in workflow systems *J. Zhejiang Univ.-Sci. A.*, **10**(8):1082-1092. [doi:10.1631/jzus.A0820366]

Michael, J., Nash, J., Keith, R., 1990. Some Conundrums Concerning Separation of Duty. Proc. IEEE Symp. on Research in Security and Privacy, p.201-209. [doi:10.1109/RISP.1990.63851]

Moffett, J.D., Sloman, M.S., 1994. Policy conflict analysis in distributed system management. *Ablex Publish. J. Organ. Comput.*, **4**(1):1-22.

Neumann, G., Strembeck, M., 2003. An Approach to Engineer and Enforce Context Constraints in an RBAC Environment. Proc. 8th ACM Symp. on Access Control Models and Technologies, p.65-79. [doi:10.1145/775412.775421]

Ni, Q., Trombetta, A., Bertino, P., Lobo, P., 2007. Privacy-Aware Role Based Access Control. Proc. 12th ACM Symp. on Access Control Models and Technologies, p.41-50. [doi:10.1145/1266840.1266848]

Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E., 1996. Role-based access control models. *IEEE Comput.*, **29**(2):38-47. [doi:10.1109/2.485845]

Simon, R., Zurko, M.E., 1997. Separation of Duty in Role Based Access Control Environments. Proc. 10th IEEE Workshop on Computer Security Foundations, p.183-194. [doi:10.1109/CSFW.1997.596811]

Sohr, K., Ahn, G.J., Gogolla, M., Migge, L., 2005. Specification and Validation of Authorization Constraints Using UML and OCL. Proc. 10th European Symp. on Research in Computer Security, p.64-79. [doi:10.1007/11555827]

Tan, E.C., Leong, P.C., Sio, L.T., 2002. Group-access control of confidential files in e-commerce management using shared-secret scheme. *Electron. Comm. Res.*, **2**(1/2):151-158. [doi:10.1023/A:1013304522599]

Traore, I., Khan, S., 2003. A Protection Scheme for Collaborative Environments. Proc. ACM Symp. on Applied Computing, p.331-337. [doi:10.1145/952532.952599]