Capability-Based Delegation Model in RBAC

Koji Hasebe Mitsuhiro Mabuchi Akira Matsushita
Graduate School of Systems and Information Engineering
University of Tsukuba
1-1-1 Tennodai, Tsukuba 305-8573, Japan
hasebe@iit.tsukuba.ac.jp, mmabu@osss.cs.tsukuba.ac.jp,
akira@cybernics.tsukuba.ac.jp

ABSTRACT

For flexible and dynamic resource management in environments where users collaborate to fulfill their common tasks, various attempts at modeling delegation of authority have been proposed using the role-based access control (RBAC) model. However, to achieve a higher level of collaboration in large-scale networked systems, it is worthwhile supporting cross-domain delegation with low administration cost. For that purpose, we propose a capability-role-based access control (CRBAC) model, by integrating a capability-based access control mechanism into the RBAC96 model. Central to this scheme is the mapping of capabilities to permissions as well as to roles in each domain, thereby realizing the delegation of permissions and roles by capability transfer. By taking this approach of capability-based access control, our model has the advantages of flexibility and reduced administration costs. We also demonstrate the effectiveness of our model by using examples of various types of delegation in clinical information systems.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access Control

General Terms

Security, Theory

Keywords

RBAC, capability-based access control, delegation

1. INTRODUCTION

Role-based access control (RBAC) is a prominent model for protection of resources in well-structured organizations such as hospitals, universities, and companies. The central idea behind this model is that users in an organization are naturally mapped to roles in the organization, and access rights are associated with the roles, thereby assigning appropriate permissions to the users belonging to these roles.

For flexible and dynamic resource management in environments where users collaborate to fulfill their common tasks, various attempts at modeling delegation of authority based on the RBAC96 model [14] have been proposed, such as RBDM [2, 3], RDM2000 [19], PBDM [21], and Atluri-Warner's model [1]. Although most studies in the literature restrict their scope to delegation in a single domain, to achieve a higher level of collaboration in distributed environments, it is worthwhile delegating across multiple domains. For example, considering the infrastructure for large-scale clinical information systems where medical doctors in different hospitals collaborate to treat their patients, delegation of access rights to electronic medical records is required across multiple hospitals.

To realize cross-domain delegation, one of the easiest ways is to prepare in advance tentative accounts for guest users in each domain. Then, by assigning suitable permissions to the guest accounts, users in different domains may access resources via these accounts. Another possible way is to register users in different domains or to use some mechanism supporting cross-domain authentication such as Kerberos [17]. However, these solutions remain unsatisfactory on the following counts. For the former solution, adequate flexibility for coping with unusual scenarios is lacking. For example, in the case of emergency in the networked clinical information system, it is useful for doctors to be able to immediately refer to a patient's medical records stored in different hospitals. However, the solution of using guest accounts would be cumbersome to deal with in such an emergency due to the need for administrator operations in assigning suitable access rights for each case. On the other hand, for the latter solution, it requires the consolidation of all the users participating in the whole network system. This may cause problems in administration costs if the network includes a large number of different domains. Moreover, to provide a high level of collaboration across different domains, especially when they independently have a hierarchy of roles based on RBAC, it is worthwhile providing a mechanism which assigns both permissions and roles of a host domain to the guest users. However, this would also be difficult in the case of networked systems that include a number of domains due to the administration costs.

To address the issue of flexible delegation based on RBAC, we propose a capability-role-based access control (CRBAC) model, by integrating a capability-based access control mechanism into the RBAC96 model. Capability-based access control mechanism (cf. [15, 10]) is considered a means for delegation of authority. In general, a capability, which is

.

an unforgeable token consisting of an object identifier and a list of permitted operations for that object. A capability represents a self-authenticating permission to access a specified object in permitted operations, whereby owners of the capability can access the object without any authentication. Moreover, it is used for cross-domain delegation by means of middleware such as HomeViews [7] or CapaFS [13].

To integrate a capability-based access control mechanism into RBAC, our central idea is to extend the RBAC96 model by introducing a set of capabilities as well as the mappings from these capabilities to permissions and roles. Thus, in our model, a user can delegate both permissions and roles by a capability transfer. By basing the approach on capability-based access control, our model has the following advantages. First, cross-domain delegation can be achieved without any authentication or specific request to the administrators. This makes flexible and smooth user-to-user delegation possible even in unusual situations such as an emergency in clinical systems. Second, by avoiding authentication, the administration cost is reduced especially in large-scale network systems. Finally, our model supports delegation of roles across different domains, which also makes it possible to assign roles in the host domain to invited users from outside. In addition, due to the mapping of capabilities to roles, appropriate permissions are automatically assigned to guest users even if the structure of roles varies.

On the other hand, since any owner of a capability has the access right specified in the capability, systems with capability-based access control has a potential security issue, namely unintended propagation of capabilities. A simple example is that a malicious participant may steal transferred capability, thus our model requires suitable authentication for each capability transfer. Moreover, it is possible that a delegatee transfers a capability to another user while a delegator does not intend to delegate authority to the user. In order to avoid such unintended propagation, in our model we provide various kinds of constraints on capabilities, such as lifetime, limitation of creation, and possible number of delegation steps. In this paper, we demonstrate the effectiveness of our model by using examples of various types of delegation in clinical information systems.

There have been a number of attempts at integration of capability-based access control mechanism with other access control models, such as Gong's identity-based capability protection system called ICAP [9], or Neuman's work [11] where Kerberos credentials were extended so that they could be used as capabilities. These studies introduced capability to realize flexible delegation and reduced administration cost. Our work has similar motivation, but we apply this approach within RBAC, which is not yet thoroughly investigated. That is the main contribution of this paper.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 introduces the CRBAC model. As with RBAC96, we first present the base model, CRBAC0, then gradually extend it by adding role hierarchy and/or constraints. We also introduce the formal definition of delegation of authority, which is represented by state transitions (i.e., rewriting instances of a model). Section 4 demonstrates how to apply our model by considering some plausible cases of clinical information systems. Finally, Section 5 concludes this paper and presents future work.

2. RELATED WORK

There have been a number of attempts at modeling delegation of authority based on the RBAC96 model. (cf. also [12] for a comprehensive guide on delegation). The RBDM0 [2] was the first attempt at modeling user-to-user delegation of roles in the RBAC model. In particular, it considered the ability of a user in a role to delegate his role membership to another user in some other role. This paper and its successor (called RBDM1) [3] also considered some extensions such as revocation, partial delegation, multi-step delegation, and delegation with role hierarchy. RDM2000 [19] was an extension of RBDM0 in supporting hierarchical roles and multi-step delegation but taking a different approach from RBDM0. This paper also introduced a rule-based language for specifying and enforcing delegation. While RBDM as well as RDM2000 considered only delegation of roles, PBDM [21] considered delegation based on both roles and permissions. PBDM also supported various types of delegation, such as user-to-user, temporal, partial, and multi-step delegation. Crampton and Khambhammettu [6] studied transfer-delegation for the RBAC model. Although the models described so far restricted their scope to delegation in a single domain, recently some models supporting cross-domain delegation have been proposed. Among these models, Atluri and Warner [1] addressed the issue of delegation in the context of workflow and presented a conditional delegation model. In addition, Gomi et al., [8] and Chadwick and Otenko [4] considered cross-domain delegation (although these works are not based on the RBAC model). The main difference between these three papers and ours is that our motivation is to support flexible delegation without any authentication mechanism. It should be noted that from a practical viewpoint, there have been many papers motivated to apply RBAC to various specific information systems. Among them, [20] as well as [16] considered clinical information systems, which were similar to those used for our example systems.

3. CRBAC

In this section, we introduce our CRBAC model. In Section 3.1, we present a brief overview of our model. In Sections 3.2 and 3.3, we define the base model, called CRBAC0, and the delegation of authority in our model. In Sections 3.4 and 3.5, we extend the basic model to CRBAC1 and 2 by introducing role hierarchy and some constraints on capabilities in a similar way to the construction of the family of RBAC96 models.

3.1 Overview

CRBAC is an extension of the RBAC96 model obtained by integrating a capability-based access control mechanism into a RBAC96 model. In a similar manner to the RBAC96 family, the CRBAC is developed from the base model (called CRBAC0) by adding the role hierarchy (CRBAC1) or various constraints (CRBAC2). (Although we also considered the consolidated model, called CRBAC3, in this paper we omit the definition because it can be obtained directly from CRBAC1 and 2).

Usually, the delegation based on the capability-based access control mechanism is achieved using the following three steps: (1) delegator (i.e., a user who wishes to delegate authority to another user) creates a capability then (2) assigns some access rights to the capability, and (3) transfer it to

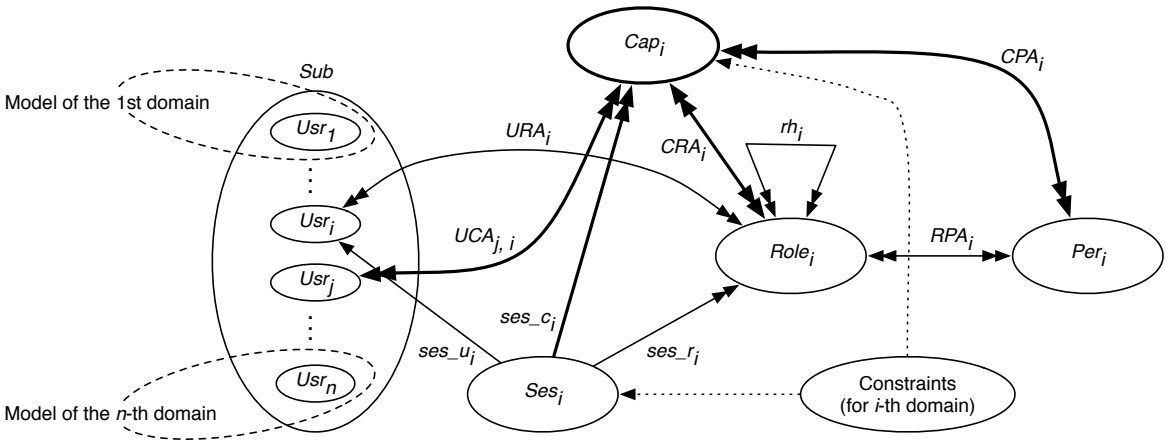


Figure 1: The family of CRBAC models

the intended delegatee (i.e., receiver of the capability).[1] To model this delegation process in the RBAC to make it possible to delegate both roles and permissions across multiple domains, we extend the RBAC96 by the following process.

First, to construct the CRBAC0 from the original RBAC0 model, we introduce some components to the RBAC0, namely, a set of capabilities, a mapping to determine the owners of the capabilities, and the assignment of roles and permissions to the capabilities. Note that in terms of these assignments we treat both roles and permissions as delegation units. Additionally, we assume that new capabilities are created either from the delegator's role or capabilities that were previously obtained. In our model, the inheritance relation with respect to the creation of capabilities is represented as a tree structure, which is grown according to the propagation of capabilities. Moreover, to model cross-domain delegation, we consider a collection of sub-models, each of which represents a single domain. On the other hand, in our model, the delegation process itself is represented as state transitions, which will be defined after introducing the CRBAC0.

Next, we extend CRBAC0 by introducing role hierarchy in a manner analogous to the RBAC96 model. We note that, in the CRBAC1, by means of the assignment of roles to capabilities, suitable permissions are automatically assigned to guest users in an environment where the host domain's role hierarchy may vary. For example, let us consider the situation where a user has a capability (say, c) to which a role (say, r) is assigned. In the RBAC96 model, the user has all the permissions assigned to r and its junior roles. Then, if the junior roles change, the permissions assigned to r also change. In our model, the permissions assigned to c are also directly reflected via the assignment of r to c.

Similar to the RBAC96 model, we also consider some constraints in the CRBAC0. Although we could consider various constraints related to users, sessions, and roles investigated in [14], we focus attention on the constraints on capability. In this paper, some constraints that would be useful to protect unintended propagation of capabilities include lifetime, creation of a new capability, and/or the number of hops of capability transfer.

[1]When implementing these steps, steps (1) and (2) are usually regarded as a single step, because in general the capability consists of an object identifier and permissions for this object. However, in our model, to categorize the types of delegation we divide this creation of capability into these two steps.

3.2 CRBAC0

The CRBAC0 is the base model of the CRBAC family. The formal definition is as follows. (See Fig 1 for a graphical presentation of the family of CRBAC models. Here, as we shall see later, the components with heavy lines are extended parts of the RBAC96 model.)

DEFINITION 1 (CRBAC0). The CRBAC0 model has the following basic components:

- Sub, Dom (sets of *subjects* and *domains*).
- $Role_i$, Per_i, Ses_i, Cap_i (sets of *roles*, *permissions*, *sessions*, and *capabilities* in the i-th domain for each $i \in Dom$). Here, Per_i may contain a special kind of permission, called *creation of capabilities*, denoted by *create*.

In addition to the above components, the following functions and relations are defined for i-th domain for each $i \in Dom$:

- $usr : Dom \rightarrow 2^{Sub}$, a function that determines the set of users in the i-th domain for each $i \in Dom$. We also use the notation Usr_i to denote $usr(i)$ and assume that $Sub = \bigcup_{i \in Dom} Usr_i$.
- $ses\_u_i : Ses_i \rightarrow Usr_i$, a function mapping each session in the i-th domain to the user who establishes it.
- $ses\_r_i : Ses_i \times 2^{Role_i}$, a function mapping each session in the i-th domain to the set of roles that is activated by this session.
- $URA_i \subseteq Usr_i \times Role_i$, a many-to-many user-to-role assignment relation.
- $RPA_i \subseteq Role_i \times Per_i$, a many-to-many role-to-permission assignment relation.

- $ses\_c_i : Ses_i \to 2^{Cap_i}$, a function mapping each session in the i-th domain to a set of capabilities.
- $UCA_{j,i} : Usr_j \to 2^{Cap_i}$, a function mapping each user in the j-th domain to a set of capabilities.
- $CRA_i \subseteq Cap_i \times Role_i$, a many-to-many capability-to-role assignment relation.
- $CPA_i \subseteq Cap_i \times Per_i$, a many-to-many capability-to-permission assignment relation.

These components satisfy the following conditions:

(C0-1) $ses\_r_i(s) \subseteq \{r | \langle ses\_u_i(s), r\rangle \in URA_i\}$, means any role activated by a session is one that is assigned to the user who establishes the session.

(C0-2) Session s has the permissions $\bigcup_{r \in ses\_r_i(s)} \{p | \langle r, p\rangle \in RPA_i\}$.

(C0-3) $ses\_c_i(s) \subseteq \{c | \langle ses\_u_i(s), c\rangle \in UCA_i\}$, means any capability activated by a session is owned by the user who establishes the session.

(C0-4) For any $r \in Role_i$ and $c \in Cap_i$, if $\langle c, r\rangle \in CRA_i$ then session s has the permissions that are determined by (C0-2) above, otherwise (i.e., $\langle c, r\rangle \notin CRA_i$) then s has the permissions $\bigcup_{c \in ses\_c_i(s)} \{p | \langle c, p\rangle \in CPA_i\}$.

As explained in the previous subsection, the CRBAC0 is a pure extension of the RBAC96 model where the components of Cap_i, ses_c_i, $UCA_{j,i}$, CRA_i, and CPA_i are the extended parts. These components play the following roles: Cap_i denotes the set of capabilities issued in the i-th domain; function ses_c_i determines the set of capabilities activated by each session; function $UCA_{j,i}$ represents the owner of capabilities; relations CRA_i and CPA_i determine which roles and permissions are assigned to each capability, respectively.

3.3 Delegation of authority

Owing to our unified treatment of roles and capabilities, the CRBAC can support various types of delegations. As mentioned in Section 3.1, the process of delegation using capability transfer can be regarded as the sequential composition of the following three basic operations:

(Step 1) *Creation.* A user creates a new capability.

(Step 2) *Assignment.* A user assigns authority to the capability.

(Step 3) *Transfer.* A user sends it to another user.

For each of the first two steps, we can consider the following two cases. For Step 1, the permission to create new capabilities is assigned to (1) a role or (2) a capability owned by the creator. For Step 2, the authority consists of (3) a set of roles or (4) a set of permissions. According these cases, we classify delegations into four types D1–D4 as shown in the following table.

Assign \ Create from	(1) Role	(2) Capability
(3) Role	D1	D3
(4) Capability	D2	D4

Furthermore, for Step 3, we classify each of D1–D4 into two categories depending on whether the delegation is made across multiple domains or not.

To provide the formal definition of the delegation of authority in our model, we first represent Steps 1 to 3 as a set of rules for the state transitions of model. (In the definition, we use the symbols $S, S', \ldots$ to denote states.) Then, by considering some sequential compositions of the transitions, we define the delegation of types D1–D4.

Before presenting the definition of delegation, to represent the relation of the capabilities with respect to the creation, we here introduce the *creation tree* as follows.

Definition 2 (Creation tree). A *creation tree* is a tree structure consisting of the following components:

- For each tree, the root node is a role in $Role_i$,
- $rc_i \supseteq Role_i \times 2^{Cap_i}$,
- $cc_i \supseteq Cap_i \times 2^{Cap_i}$.

For readability, throughout we use the infix notations $\to_{rc_i}$ and $\to_{cc_i}$ to denote rc_i and cc_i, respectively. Intuitively, $\langle r, c\rangle \in rc_i$ and $\langle c_1, c_2\rangle \in cc_i$ represent that capability c and c_2 are created from role r and capability c_1, respectively. In such a case, we often say "r (c_1, resp.) is the *parent* of c (c_2, resp.)". The reflexive asymmetric transitive closure of cc_i is denoted by cc_i^*. Additionally, we use the notation CT_i to denote $rc_i \cup cc_i^*$, thus it determines the creation relation of the capabilities in i-th domain.

The relations rc_i and cc_i satisfy the following conditions for each $i \in Dom$.

(C0-5) If $r \to_{rc_i} c$ then $\langle r, create\rangle \in RPA_i$.

(C0-6) If $c \to_{cc_i}^* c'$ then $\langle c, create\rangle \in CPA_i$.

(C0-7) If $r \to_{rc_i} c$ then
$\{p | \langle r, p\rangle \in RPA_i\} \supseteq \{p | \langle c, p\rangle \in CPA_i\}$.

(C0-8) If $c \to_{cc_i}^* c'$ then
$\{p | \langle c, p\rangle \in CPA_i\} \supseteq \{p | \langle c', p\rangle \in CPA_i\}$.

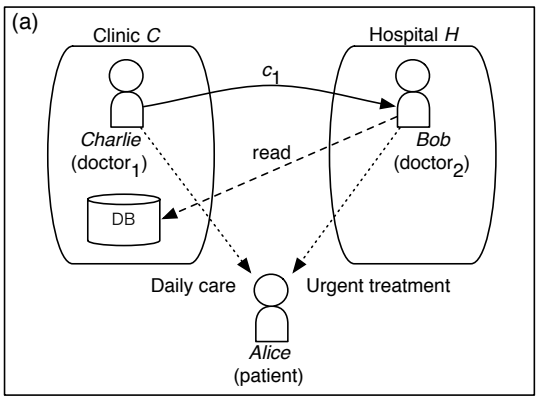
Intuitively, the former two represent the requirement that a new capability can be created whenever the right of creation is assigned to the parent. On the other hand, the latter two represent the requirement that the permissions assigned to a created capability are weaker than the those assigned to the parent.

Definition 3 (Basic operations for delegation). The basic operations for delegation (i.e., *creation*, *assignment*, and *transfer*) are defined by the following rules of state transitions for a model.

Creation rule: Let S be a state in which $u \in Usr_i$, $c \in Cap_i$.

(1) If $r \in Role_i$ with $\langle u, r\rangle \in URA_i$ and $\langle r, create\rangle \in RPA_i$ in S, then this rule can be applied to S and defines new state S' by replacing Cap_i and CT_i in S with Cap_i' and CT_i', respectively, such that:

- $Cap_i' := Cap_i \cup \{c'\}$ where $c' \notin Cap_i$;
- $CT_i' := CT_i \cup \langle r, c\rangle$.



(b) State S' (after the delegation)

	Clinic C	Hospital H
Usr	$Charlie$	Bob
$Role$	$doctor_1$	$doctor_2$
Per	$create$, $may(DB, read)$, $may(DB, write)$	
Cap	$[c_1]$	
URA	$\langle Charlie, doctor_1\rangle$	$\langle Bob, doctor_2\rangle$
RPA	$\langle doctor_1, create\rangle$, $\langle doctor_1, may(DB, read)\rangle$, $\langle doctor_1, may(DB, write)\rangle$	
UCA		$[\langle Bob, \{c_1\}\rangle]$
CPA	$[\langle c_1, may(DB, read)\rangle]$	
CT	$[doctor_1 \rightarrow c_1]$	

Figure 2: Example 1

(2) If $p \in Per_i$ with $\langle u, c\rangle \in UCA_i$ and $\langle c, create\rangle \in CPA_i$, then this rule can be applied to S and defines new state S' by replacing Cap_i and CT_i in S with Cap'_i and CT'_i, respectively, such that:

- $Cap'_i := Cap_i \cup \{c'\}$ where $c' \notin Cap_i$;
- $CT'_i := CT_i \cup \langle c, c'\rangle$.

The state transitions of these types are denoted by $S \overset{cre(u,r,c')}{\rightsquigarrow} S'$ (for Case (1)) and $S \overset{cre(u,c,c')}{\rightsquigarrow} S'$ (for Case (2)), respectively, and we can say"capability c' is created by user u from his role r (from his capability c, resp.)"

Assignment rule: Let S be a state in which $u \in Usr_i$ and c' $(\in Cap_i)$ is created by u.

(3) If $R' \supseteq R \supseteq Role_i$ in S such that $\langle u, r\rangle \in URA_i$ for any $r \in R$, then this rule can be applied to S and defines new state S' by replacing CRA_i in S with $CRA'_i := CRA_i \cup \{\langle c', r\rangle | r \in R'\}$.

(4) If $P \supseteq Per_i$ in S, then this rule can be applied to S and defines new state S' by replacing CPA_i in S with $CPA'_i := CPA_i \cup \{\langle c', p\rangle | p \in P\}$.

State transitions of these types are denoted by $S \overset{asg(R',c')}{\rightsquigarrow} S'$ (for Case (3)) and $S \overset{asg(P',c')}{\rightsquigarrow} S'$ (for Case (4)), respectively, and we can say "roles R' (permissions P', resp.) is assigned to capability c".

Transfer rule: Let S be a state in which $u \in Usr_i$, $u' \in Usr_j$, $c' \in Cap_i$, and c' is created by u (where i and j may be the same domain number). Then this rule can be applied to S and defines new state S' by replacing $UCA_{i,j}$ in S with $UCA'_{i,j} := UCA_{i,j} \cup \{\langle u', c'\rangle\}$. State transitions of this type are denoted by $S \overset{trans(u,u',c')}{\rightsquigarrow} S'$ and we can say "capability c' is transferred from user u to the other user u'". In particular, such transfer is called a *cross-domain transfer* if $i \neq j$.

By combining these basic operations, we define the delegations of types (D1) to (D4).

Definition 4 (Delegation). The delegations of types (D1) to (D4) are sequential compositions of basic operations *Creation*, *Assignment*, and *Transfer* as follows (where ";" is used to denote the sequential-composition operator):

- (D1): $S \overset{cre(u,c',r)}{\rightsquigarrow}; \overset{asg(R',c')}{\rightsquigarrow}; \overset{trans(u,u',c')}{\rightsquigarrow} S'$.
- (D2): $S \overset{cre(u,c',r)}{\rightsquigarrow}; \overset{asg(P',c')}{\rightsquigarrow}; \overset{trans(u,u',c')}{\rightsquigarrow} S'$.
- (D3): $S \overset{cre(u,c',c)}{\rightsquigarrow}; \overset{asg(R',c')}{\rightsquigarrow}; \overset{trans(u,u',c')}{\rightsquigarrow} S'$.
- (D4): $S \overset{cre(u,c',c)}{\rightsquigarrow}; \overset{asg(P',c')}{\rightsquigarrow}; \overset{trans(u,u',c')}{\rightsquigarrow} S'$.

For readability, we also use the following notations to denote delegations of these types:

- (D1): $S \overset{D1(u,u',c',r,R')}{\rightsquigarrow} S'$.
- (D2): $S \overset{D2(u,u',c',r,P')}{\rightsquigarrow} S'$.
- (D3): $S \overset{D3(u,u',c',c,R')}{\rightsquigarrow} S'$.
- (D4): $S \overset{D4(u,u',c',c,P')}{\rightsquigarrow} S'$.

Finally, to help the readers understand the formal definitions we provide a simple example of a scenario in a clinical information system. (See Fig 2 (a) for the graphical presentation). Larger and more realistic examples are presented in Section 5.

Example 1. One day, Alice suddenly loses consciousness and is transferred to emergency hospital H. The result of the urgent examination indicates that she is in hypoglycemic shock. Bob, a doctor in the hospital, immediately starts to treat Alice and looks for her primary care doctor to refer to her medical record, because he suspects she has severe diabetes mellitus on the basis of her symptoms. Subsequently, from her ID card, Bob identifies her primary care doctor as Charlie in local clinic C. Charlie then allows Bob to refer to the medical record stored in a database of the clinic by creating new capability c_1 and assigning permission $may(DB, r)$ (i.e., permission to read the contents of the database) and passes it on to Bob.

Let us consider that the delegation is done by D2, that is, Charlie creates the new capability c_1 by activating his role $doctor_1$ and assigns only the permission to read the contents of a database. In this case, the basic components are as shown in the table of Fig 2 (b), where the columns list the

components of clinic C or hospital H. (We omit some components if they do not explicitly appear in this instance.) In this table, the items surrounded by square brackets indicate the new ones which are added by the delegation of c_1. The expression "$may(o, r)$" represents the permission to access the object o with a certain right r (e.g. read, write, or execution).

More precisely, the delegation process can be described by the following state transitions. First, by the creation of rule $cre(Charlie, c_1, doctor_1)$, we obtain a new state by respectively replacing Cap_C and CT_C (components of the initial state) with:

- $Cap'_C := Cap_C \cup \{c_1\}$,
- $CT'_C := CT_C \cup \langle doctor_1, c_1 \rangle$.

Then by the assignment rule $asg(may(DB, read), c_1)$, we obtain a new state by replacing CPA_C with:

- $CPA'_C := \langle c_1, may(DB, read) \rangle$.

Finally, by the transfer rule $trans(Charlie, Bob, c_1)$, we obtain a new state by replacing UCA_C with:

- $UCA'_C := UCA_C \cup \langle Bob, \{c_1\} \rangle$.

Moreover, the activation of capability c_1 by Bob to read the clinical record can be represented by the state transition:

- $Ses'_C := Ses_C \cup \{s\}$

(where Ses_C is the set of sessions in the adjacent state) with

- $ses\_u'_C(s) = Bob$,
- $ses\_c'_C(s) = \{c_1\}$.

Remark 1. So far we explained the base model and the delegation in CRBAC. We discuss here how this model can be implemented. A possible way is to implement RBAC permissions using access control list (ACL) approach and the capability-based permissions are implemented by extension of the ACL approach. That is, the i-th domain (for each i) has a server that manages RBAC permissions by ACL-based access control mechanism with two matrices specifying URA_i and RPA_i. When a user in the j-th domain accesses an object in the i-th domain by activating his capability (say, c), the server checks the validity of this capability then dynamically adds a tentative user, whose name is just specified as "guest", to the matrix of URA_i. If some roles are assigned to c, then these roles are assigned to this tentative user in the matrix of URA_i, thereby the guest user may activate these roles. On the other hand, if some permissions are assigned to c, then the specific role, called r^*, is assigned to the tentative user in the matrix of URA_i, while a list which specifies that all the permissions of c are assigned to r^* is also added to the matrix of RPA_i, thereby the guest user has all the permissions. (Note that r^* plays a role in associating the guest user with the permissions.) Then, after closing the session, the server removes both the tentative user and the corresponding permissions from the matrices. In other words, in our model, capability plays a role of token that invokes such an automatic revision of ACL instead of asking the administrator of such a revision.

Actually, this dynamic revision of ACLs would burden the server, especially when the number of objects specified in a capability is large. However, this mechanism makes it possible to avoid the administrator operations and authentication, thus in the case of emergency, users may delegate authority without sending the password of the prepared guest user account and without any request to the administrator.

3.4 CRBAC1

Based on the CRBAC0 model, we develop an extended model named CRBAC1 by introducing role hierarchy. This extension is essentially the same as for RBAC1. That is, CRBAC1 is defined as follows:

Definition 5 (CRBAC1). The CRBAC1 model consists of the following components:

- Sub, Rig, Dom, and usr are unchanged from CRBAC0.
- The sets $Role_i$, Cap_i, Obj_i, Ses_i, Per_i, ses_u_i, ses_r_i, URA_i, $UCA_{i,j}$, RPA_i, CPA_i, and CRA_i are also defined as in the CRBAC0 for each domain i.
- rh_i, a partial order over $Role_i$, called *role hierarchy*. (The infix notation $r' \geq_{rh_i} r$ is also used to denote role hierarchy and we can say "r' is senior role of r" or "r is junior role of r'" in the same sense as in the RBAC96 model.)

In addition to the above setting, the following conditions are satisfied for each i.

(C1-1) $ses\_r_i(s) \subseteq \{r \mid \exists r' \geq_{rh_i} r(\langle ses\_u_i(s), r' \rangle \in URA_i)\}$.

(C1-2) For each session $s \in Ses_i$, it has the set of permissions $\bigcup_{r \in ses\_r_i(s)} \{p \mid \exists r'' \leq_{rh_i} r(\langle r'', p \rangle \in RPA_i)\}$.

(C1-3) For each session $s \in Ses_i$, it has the set of permissions $\bigcup_{r' \in \{r \mid \langle c_i, r \rangle \in CRA_i \wedge c_i \in ses\_c_i(s)\}} \{p \mid \exists r'' \leq r' (\langle r'', p \rangle \in RPA_i)\}$

The former two are related to the role hierarchy, that are the same as in the RBAC96. Intuitively, the condition (C1-1) means the requirement that every role activated by session s is less or equally powerful (junior) to any role of a user who establishes the session s. (C1-2) means the requirement that the permissions in session s are those directly assigned to the session's roles and all of their junior roles. On the other hand, the third condition guarantees the inheritance of permissions with respect to the role hierarchy. In other words, (C1-3) means that if a role is assigned to a capability and it is activated, then all the permissions are assigned to this role and to all the junior ones. More specifically, we explain this by using a slightly extended version of Example 1 as follows:

Example 2. After urgent treatment for Alice (described in Example 1), Bob is invited to clinic C to treat Alice in collaboration with Charlie. For that purpose, Charlie delegates to $doctor_1$ via the capability c_2. In clinic C, the role $doctor_1$ has a $nurse$ as a junior role, so Bob may have all the permissions assigned to the $nurse$ by activating c_2. One day, the role structure in clinic C changes, so $doctor_1$ also has a *technician* as a junior role.

The state (say, S'') after the delegation of c_2 can be specified by taking RPA''_C and CRA''_C as follows and all the other components as the table in Fig 2. (Here p_1 and p_2 are certain permissions.)

- $RPA''_C := \{\langle doctor_1, p_1\rangle, \langle nurse, p_2\rangle\}$.
- $CRA''_C := \{\langle c_2, doctor_1\rangle\}$.

For this state, when Bob activates c_2 by session s_2 (i.e., $ses\_c_i(s_2) = \{c_2\}$ and $ses\_u_i(s_2) = Bob$), by condition (C1-3) he has the permissions p_1 and p_2. (Note that, in this case, the variable r' in (C1-3) ranges over the singleton $\{doctor_1\}$.) Again, after adding the new role *technician* (with $\langle technician, p_3\rangle \in RPA''_C$), when Bob activates c_2, by condition (C1-3) he also has permission p_3 in addition to p_1 and p_2.

As this example indicates, this inheritance is useful for environments where users in different domains collaborate on a specific domain performing as roles in this domain and the role hierarchy may vary.

3.5 CRBAC2

Similar to the development process of the RBAC96 family, we here develop another extension named CRBAC2. In this paper, we focus attention on the constraints related to the capability. In particular, we consider the following five constraints, that would be useful for prohibiting unintended propagation of capabilities.

(R1) Lifetime

(R2) The number of activations

(R3) The number of creations of new capabilities

(R4) The inheritance of permissions related to role hierarchy

(R5) The number of hops of capability transfer

There are also other constraints such as the depth of inheritance of permission to create new capabilities. On the other hand, here we do not consider the constraints on the destination of capability transfer, because it is not plausible from the viewpoint of the implementation.

From now on, we present the formal definitions of constraints R1–R5 below:

R1 (*Lifetime*): To make the discussion simpler, we assume that all the users as well as all the system components (e.g., terminal computers) refer the unique global clock. To formalize the notion of lifetime, we first introduce the following constant and function:

- *time* ($\in \mathbb{Z}^+$), the current time of the global clock represented by non-negative integer;
- $life_i : Cap_i \to \mathbb{Z}^+ \times \mathbb{Z}^+$, a function mapping each capability in the i-th domain to the time from which it can be activated and the time from which it expires.

Note that we may omit the onset and expiry time by setting them as 0 and $+\infty$, respectively.

By means of these constants and functions, the lifetime of a capability can be represented by the following conditions for any $c \in Cap_i$ and for any $i \in Dom$:

- $fst(life(c)) < time \Rightarrow ses\_c_i(c) \notin Ses_i$.
- $snd(life(c)) > time \Rightarrow ses\_c_i(c) \notin Ses_i$.

Here *fst* and *snd* are the first and second projection of the input.

R2 (*The number of activation*): To define this constraint we first introduce the following functions:

- $lim\_act_i : Cap_i \to \mathbb{N}$;
- $cnt\_act_i : Cap_i \to \mathbb{N}$.

The former determines the possible number of activations, while the latter counts the number of activations so far. Thus, by using these functions, this constraint can be represented as the following condition for any $c \in Cap_i$ and for any $i \in Dom$.

$$lim\_act_i(c) \geq cnt\_act_i(c).$$

R3 (The number of creations): For a constraint of this kind, there are two options: the number of creations and the depth of descending from a single capability. In either case, we can represent these constraints as the restrictions on the width and depth of the creation tree, respectively. That is, we first introduce the following functions:

- $lim\_cre\_w_i : Cap_i \to \mathbb{N}$;
- $lim\_cre\_d_i : Cap_i \to \mathbb{N}$.

Then the constraints are represented by the following conditions for any $c \in Cap_i$ and for any $i \in Dom$:

- $lim\_cre\_w_i(c) \geq |\{c' | c \to_{cc_i} c' \wedge c \in Cap_i\}|$;
- There is no sequence of capabilities $c_1, c_2, \ldots c_k$ with $k = lim\_cre\_d_i(c)$ such that $c \to_{cc_i} c_1 \to_{cc_i} \cdots c_{k-1} \to_{cc_i} c'$.

R4 (*The inheritance of permissions*): Although there are various constraints of this kind, they can be defined in common by some modification of condition (C1-3) introduced in the previous subsection. One of the simplest examples is the constraint prohibiting any inheritance of permissions of junior roles that can be defined by replacing (C1-3) with the following condition:

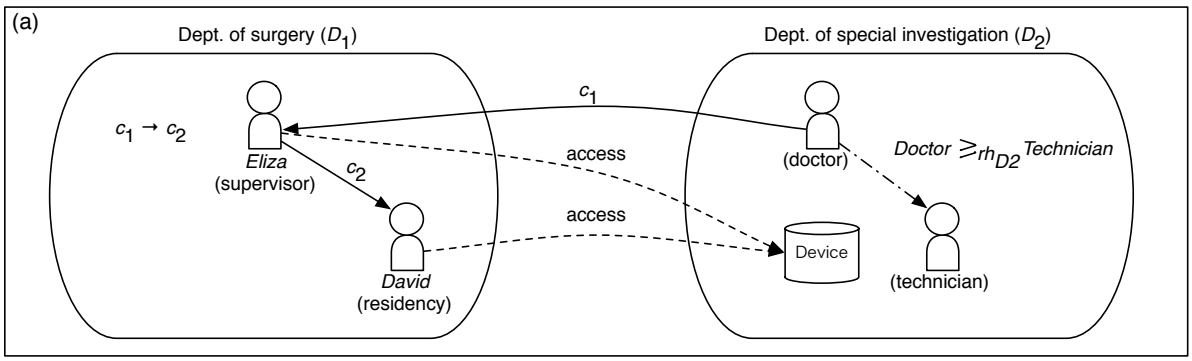
(C1-3′) For each session $s \in Ses_i$, it has the set of permissions $\bigcup_{r' \in \{r \mid \langle c_i, r\rangle \in CRA_i \wedge c_i \in ses\_c_i(s)\}} \{p \mid (\langle r', p\rangle \in RPA_i)\}$.

R5 (*The number of hops of capability transfer*): To define this, we first introduce the following functions:

- $lim\_hop_i : Cap_i \to \mathbb{N}$, a mapping of each capability to the limited number of times of capability transfer;
- $cnt\_hop_i : Cap_i \to \mathbb{N}$, a mapping of each capability to the number of the capability transfers that have been executed so far.

As a formalization of R2, the former determines the possible number of hops of capability transfer, while the latter counts the number of hops so far. Thus, the constraint is represented as the following condition for any $c \in Cap_i$ and for any $i \in Dom$:

$$lim\_hop_i(c) \geq cnt\_hop_i(c).$$



(b) State S'' (after the delegation)

	Dept. of Surgery (D_1)	Dept. of SI (D_2)
Usr	$David$, $Eliza$	$Doctor$, $Technician$
$Role$	$supervisor$, $residency$	$doctor$, $technician$
Per		$create$, $may(Dev, operate)$, $may(Dev, setup)$
Cap	$[c_2]$	$[c_1]$
URA	$\langle David, residency\rangle$, $\langle Eliza, supervisor\rangle$	$\langle Doctor, doctor\rangle$, $\langle Technician, technician\rangle$
RPA		$\langle doctor, create\rangle$, $\langle doctor, may(Dev, operate)\rangle$, $\langle technician, may(Dev, setup)\rangle$
UCA	$[\langle Eliza, \{c_1\}\rangle, \langle David, \{c_2\}\rangle]$	
CPA	$[\langle c_2, may(Dev, operate)\rangle]$	
CRA	$[\langle c_1, doctor\rangle]$	
CT	$[c_1 \rightarrow c_2]$	$[doctor \rightarrow c_1]$

Figure 3: Case 1

The constraints presented above are indeed useful to avoid security issues with respect to delegation based on the capability transfer. For example, considering the scenario of Example 1, it would be better to restrict the lifetime (i.e. R1) and/or the number of activations (i.e. R2) of capability c_1, and it is also the case for constraints R3 and R5 in prohibiting unwanted propagation of the permission in hospital H. Additionally, for the case of Example 2, if Bob does not need to activate roles other than $doctor_1$ for collaboration purposes, it would be better to cut off the inheritance relationship of $doctor_1$ by the constraint R4.

4. CASE STUDY

In this section we demonstrate how to apply our model to some specific environments. For this purpose we consider two cases of clinical information systems.

4.1 Case 1

The first case is a scenario where delegations are done both in a single domain and across multiple domains. (See also Fig 3 (a) for the graphical presentation of this case.) This case also includes the creation of capabilities both from a role and another capability. We also consider some constraints on the capabilities.

Case 1. David starts his new post of residency in the department of surgery in hospital H, where Eliza is his supervisory doctor. In this hospital, Eliza has charge of a special investigation, so she has the capability (called c_1), which is created from role *doctor* by a doctor in the department of special investigation, to operate the computer device for the investigation. In the department of special investigation, other than doctors, there are some technicians who engage in the setup of this device. So, they are assigned the role *technician* to permit only the setup of the device. Two months later, David has gained sufficient training to handle this special device, thus Eliza creates a new capability (called c_2) from c_1 to operate the device and passes it to him. Here, the lifetime of c_2 is set as his residency period.

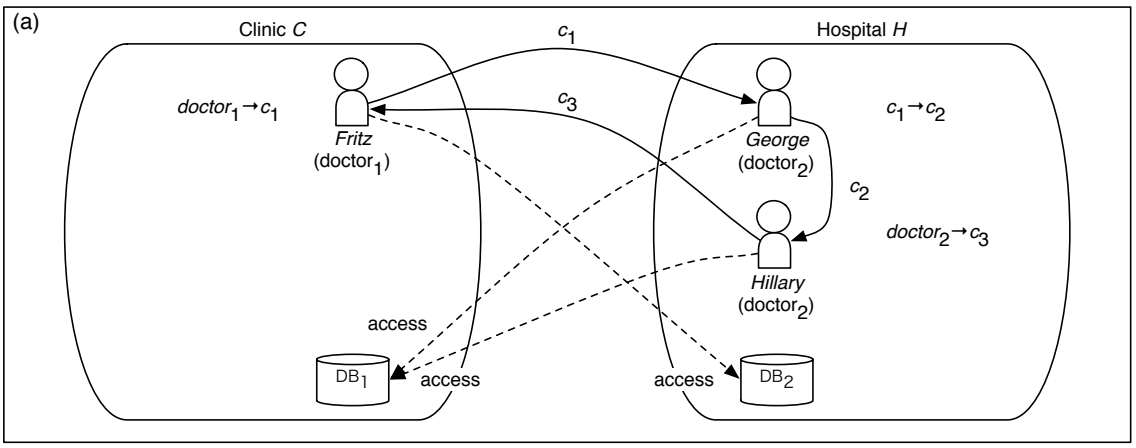
The basic components of the state after the delegations of c_1 and c_2 are presented in the table of Fig 3 (b), where the components surrounded by square brackets are newly added by the delegation process.

The delegation process can be specified by the following state transitions:

- $S \rightsquigarrow S'$ by
  $D1(Doctor, Eliza, c_1, doctor, \{doctor\})$;
- $S' \rightsquigarrow S''$ by
  $D4(Eliza, David, c_2, c_1, \{may(Dev, operate)\})$.

Here, S, S', and S'' are initial state, the state after the first delegation, and the state after the second delegation. More precisely, according to the D1 rule, the intermediate state S' can be obtained by the following replacement:

- $Cap'_{D_2} := Cap_{D_2} \cup \{c_1\}$;
- $CT'_{D_2} := CT_{D_2} \cup \{\langle doctor, c_1\rangle\}$;
- $UCA'_{D_2} := UCA_{D_2} \cup \{\langle Eliza, \{c_1\}\rangle\}$;



(b) State S''' (after the delegation)

	Clinic C	Hospital H
Usr	$Fritz$	$George$, $Hillary$
$Role$	$doctor_1$	$doctor_2$
Per	$create$, $may(DB_1, access)$	$create$, $may(DB_2, access)$
Cap	$[c_1, c_3]$	$[c_2]$
URA	$\langle Fritz, doctor_1\rangle$	$\langle George, doctor_2\rangle$, $\langle Hillary, doctor_2\rangle$
RPA	$\langle doctor_1, create\rangle$, $\langle doctor_1, may(DB_1, access)\rangle$	$\langle doctor_2, create\rangle$, $\langle doctor_2, may(DB_2, access)\rangle$
UCA	$[\langle Fritz, \{c_3\}\rangle]$	$[\langle George, \{c_1\}\rangle, \langle Hillary, \{c_2\}\rangle]$
CPA	$[\langle c_1, create\rangle, \langle c_1, may(DB, access)\rangle]$	$[\langle c_2, may(DB, access)\rangle]$
CT	$[doctor_1 \rightarrow c_1]$	$[c_1 \rightarrow c_2, doctor_2 \rightarrow c_3]$

Figure 4: Case 2

- $CRA'_{D_2} := CRA_{D_2} \cup \{\langle c_1, doctor\rangle\}$.

Again, according to the D4 rule, the final state S' can be obtained from S_1 by the following replacement:

- $Cap''_{D_1} := Cap'_{D_1} \cup \{c_2\}$;
- $CT''_{D_1} := CT'_{D_1} \cup \{\langle c_1, c_2\rangle\}$;
- $UCA''_{D_1} := UCA'_{D_1} \cup \{\langle David, \{c_2\}\rangle\}$;
- $CPA''_{D_1} := CPA'_{D_1} \cup \{\langle c_2, \{may(DB, operate)\rangle\}$.

In this case we may consider the following constraints. First, with the delegation of c_1, Eliza may operate the computer device by activating the role *doctor*. Moreover, in domain D_2, if *doctor* is in a senior role to the *technician* then she is also allowed to set up the device by the condition (C1-3) in either CRBAC1 or CRBAC3. Otherwise, if Eliza does not need to set up the device, it is possible to introduce constraint R4 to forbid her to do so. In addition to the above constraint, Eliza can weaken the authority of c_2 by adding lifetime or by disabling any new creation or transfer using R1, R3, R5, respectively. We note that such constraints can be naturally represented in our model by using some special functions introduced in Section 3.5.

4.2 Case 2

The second case is a more complex scenario. (See also Fig 4 (a) for the graphical presentation of this case.)

Case 2. Alice regularly sees Fritz, her primary care doctor at a clinic C. One day, Alice's medical condition takes a turn for the worse, so he puts Alice in touch with George, a doctor for cardiovascular internal medicine in the general hospital H, to provide an examination by a specialist. At this time, to allow George to read Alice's medical record in the clinic, Fritz creates a new capability (called c_1) from his role $doctor_1$ and passes it to him. Then he performs physical examinations referring to Alice's past medical record. As a result, George considers that she requires hospitalization to undergo cardiac surgery. He consults Hillary, a doctor in the department of cardiac surgery and allows her to read Alice's medical record by making another capability (called c_2) from this capability c_1. Hillary agrees with George's assessment and admits Alice to hospital H. Then Hillary makes a plan for diagnoses and treatments based on the examinations and the past medical record. In addition, Fritz is accepted as a care member for Alice by the regional referral system, so Hillary makes a capability (called c_3) to access Alice's medical chart in Hospital H, and passes it to him.

The basic components of the final state are presented in the table of Fig 4 (b). The delegation process can be specified by the following state transitions:

- $S \rightsquigarrow S'$ by
  $D2(Fritz, George, c_1, doctor_1,$
  $\{create, may(DB_1, access)\})$
- $S' \rightsquigarrow S''$ by
  $D4(George, Hillary, c_2, c_1, \{may(DB_1, access)\})$

- $S'' \rightsquigarrow S'''$ by
  $D2(Hillary, Fritz, c_3, c_2, \{may(DB_2, access)\})$

Similar to the previous case, here we may consider some constraints on the delegation. First, when Fritz creates c_1, he can set the limit number of creations (i.e., R3) as well as the number of hops of capability transfer (i.e., R5). So, if he wants to forbid any propagation of the permission assigned to c_1, he can set the numbers of both $lim\_hop_C(c_1)$ and $lim\_cre\_w_C(c_1)$ as 0. In this case, George cannot propagate it because it would violate the conditions of R3 and R5.

In closing the case study, we would like to stress the following points. First, as these scenarios indicate, our model supports a flexible user-to-user delegation of both roles and permissions without any authentication. This would be helpful for collaboration between different domains. Next, as presented in the explanations, our model provides delegators to set various constraints on the propagation of capability by combining basic constraint rules R1–R5. This would be useful to compensate for the intrinsic weakness in the capability-based access control.

5. CONCLUSIONS AND FUTURE WORK

We proposed an access control model named CRBAC integrating a capability-based access control mechanism into the RBAC96 model. Owing to the flexibility of capability-based access control, our model supported both delegation of roles and permissions in terms of capability transfer. Additionally, CRBAC made possible cross-domain delegation without any authentication, which should reduce the administration costs. Similar to the RBAC96 model, we also addressed some extensions by introducing role hierarchy as well as various types of constraints on the capability-based delegation, thereby providing a high level of collaboration between different organizations while preventing unintended propagation of the capabilities.

For future work, one of the most interesting and worthwhile directions is to develop a formal method for security verification of the CRBAC. In particular, we are interested in a method for detecting unintended propagation of capability by searching all possible delegation processes using a model-checking technique [5]. We are also interested in the feasibility of the proposed model, which could be investigated through a prototype implementation.

6. ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for useful suggestions and comments which help us improve this paper. This study was supported in part by the Grant-in-Aid for the Global COE Program on "Cybernics: fusion of human, machine, and information systems" at the University of Tsukuba.

7. REFERENCES

[1] V. Atluri and J. Warner. Supporting conditional delegation in secure workflow management systems, *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT'05)*, pp.49-58, 2005.

[2] E. Barka and R. Sandhu. A Role-based Delegation Model and Some Extensions. *Proceedings of the 23rd National Information Systems Security Conference*, pp.101-114, 2000.

[3] E. Barka and R. Sandhu. Role-Based Delegation Model/Hierarchical Roles, *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC' 04)*, pp.396-404, 2004.

[4] D. W. Chadwick and A. Otenko. The PERMIS X.509 Role Based Privilege Management Infrastructure, *Proceedings of the 10th IFIP Open Conference on Communications and Multimedia Security (CMS'06)*, pp.67-86, 2006.

[5] E. M. Clarke, O Grumberg, and D. A. Peled. *Model Checking*, MIT Press, 2000.

[6] J. Crampton and H. Khambhammettu. Delegation in Role-Based Access Control, *Proceedings of the 11th European Symposium on Research in Computer Security (ESORICS'06)*, LNCS vol.4189, pp.174-191, 2006.

[7] R. Geambasu, M. Balazinska, S. D. Gribble, and M. Levy. HomeViews: Peer-to-Peer Middleware for Personal Data Sharing Applications, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.235-246, 2007.

[8] H. Gomi, M. Hatakeyama, S. Hosono, and S. Fujita. A Delegation Framework for Federated Identity Management, *Proceedings of the ACM Workshop on Digital Identity Management (DIM'05)*, pp.94-103, 2005.

[9] L. Gong. A Secure Identity-Based Capability System, *IEEE Symposium on Security and Privacy*, pp.56-63, 1989.

[10] H. M. Levy. *Capability-Based Computer Systems*, Digital Equipment Corporation, 1984.

[11] B. C. Neuman. Proxy-Based Authorization and Accounting for Distributed Systems, *Proceedings of the 13th International Conference on Distributed Computing Systems*, pp.283-291, 1993.

[12] Q. Pham, J. Reid, A. McCullagh, and E. Dawson. On a Taxonomy of Delegation, *the FIFP International Information Security Conference (IFIP/SEC-2009)*, pp.353-363, 2009.

[13] J. T. Regan and C. D. Jensen. Capability File Names: Separating Authorization from User Management in an Internet File System, *Proceedings of the USENIX Security Symposium*, pp.211-233, 2001.

[14] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models, *IEEE Computer*, vol.29, no.2, pp.38-47, 1996.

[15] L. Snyder. Formal Models of Capability-Based Protection Systems, *IEEE Trans. on Computers*, vol.c-30, no.3, pp.172-181, 1981.

[16] K. Sohr, M. Drouineaud, and G-J. Ahn. Formal Specification of Role-based Security Policies for Clinical Information Systems, *Proceedings of the 20th Annual ACM Symposium on Applied Computing (SAC'05)*, pp.332-339, 2005.

[17] J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An Authentication Service for Open Network Systems, *Proceedings of the Winter 1988 USENIX Conference*, pp.191-201, 1988.

[18] J. Wainer and A Kumar. A Fine-grained, Controllable, User to User Delegation Method in RBAC, *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT'05)*, pp.59-66, 2005.

[19] L. Zhang, G. Ahn, B. T. Chu. A rule-based framework for role based delegation, *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT'01)*, pp.153-162, 2001.

[20] L. Zhang, G. Ahn, and B. T. Chu. A Role-Based Delegation Framework for Healthcare Information Systems, *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT'02)*, pp.125-134, 2001.

[21] X. Zhang, S. Oh, and R. Sandhu. PBDM: A Flexible Delegation Model in RBAC, *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT'03)*, pp.149-157, 2003.