

Managing Delegation in Access Control Models

Meriam Ben Ghorbel-Talbi^{a,b}, Frederic Cuppens^a, Nora Cuppens-Boulahia^a, Adel Bouhoula^b

^aGET/ENST Bretagne,

2 rue de la Chataigneraie, 35512, Cesson Sevigne Cedex, France

{meriam.benghorbel,frederic.cuppens,nora.cuppens}@enst-bretagne.fr

^bSUP'COM Tunis,

Route de Raoued Km 3.5, 2083 Ariana, Tunisie

bouhoula@planet.tn

Abstract

In the field of access control, delegation is an important aspect that is considered as a part of the administration mechanism. Thus, a complete access control must provide a flexible administration model to manage delegation. Unfortunately, to our best knowledge, there is no complete model for describing all delegation requirements for role-based access control. Therefore, proposed models are often extended to consider new delegation characteristics, which is a complex task to manage and necessitate the redefinition of these models.

In this paper we describe a new delegation approach for extended role-based access control models. We show that our approach is flexible and is sufficient to manage all delegation requirements.

1. Introduction

The delegation is the process whereby a user without any administrative prerogatives obtains the ability to grant some authorizations. In the field of access control, defining concept of delegation is a difficult issue and only few works are dedicated to this point [2, 3, 4, 12, 13].

These works showed that delegation is a complex problem to solve and is generally modeled separately from other administration requirements. The reason is that proposed models are generally based on the RBAC model [9] (Role-Based Access Control), which is not expressive enough to deal with delegation requirements such as temporary, partial, multi-step or multiple delegation. For instance, in the RBAC model, the only way to grant permission to a subject is by granting this permission to a given role and then assigning this role to the subject. This is a rigid scenario, which does not adequately answer the need of fine-grained

delegation.

Therefore, it is necessary to extend the RBAC model by adding new components, such as new types of roles, permissions and/or relations. This is a complex task to manage, and to our best knowledge, there is no complete model for describing all delegation requirements. Thus, delegation models themselves are extended to consider new delegation characteristics.

In this paper, we aim at proposing a flexible and complete delegation approach for role based access control. Our work is based on the OrBAC [1] (Organization based Access Control) formalism, which provides integrated framework to deal with various security requirements including delegation requirements.

Namely, the OrBAC model gives means to specify contextual authorizations, which facilitate the modeling of delegation characteristics such as temporary delegation, cascading revocation, etc. In addition, in AdOrBAC [7] (Administration model for OrBAC) a large number of conditions can be expressed thanks to the use of views (in OrBAC model a view is an access control entity used to put together objects to which apply the same authorizations, for more details see [7]).

This provides means to specify fine-grained delegation constraints, such as prerequisite conditions for the grantor (the user who performs the delegation) and the grantee (the user who receives the delegation). Therefore the administrator can restrict the delegation by specifying delegation constraints that grantor and grantee must satisfy.

This paper is organized as follows. In section 2 we start with basic concepts of the OrBAC and AdOrBAC models. In section 3 we present our delegation model. We discuss in section 4 the complexity and the decidability of our approach. Then related work are given in section 5. Finally, concluding remarks are made in section 6.

2. Basic concepts for OrBAC

Before presenting our delegation model, we shall briefly recall the main components of OrBAC.

2.1. OrBAC model

The central entity in OrBAC is the entity organization. Intuitively, an organization is any entity that is responsible for managing a security policy.

The objective of OrBAC is to specify the security policy at the organization level that is independently of the implementation of this policy.

Thus, instead of modeling the policy by using the concrete concepts of subject, action and object, the OrBAC model suggests reasoning with the roles that subjects, actions or objects play in the organization.

The role of a subject is simply called a role, whereas the role of an action is called activity and the role of an object is called view.

In OrBAC, there are eight basic sets of entities: Org (a set of organizations), S (a set of subjects), A (a set of actions), O (a set of objects), R (a set of roles), \mathbb{A} (a set of activities), V (a set of views) and C (a set of contexts). However, for the sake of simplicity, we assume that the policy applies to a single organization and thus we omit the Org entity in the following.

In the following we present the basic OrBAC builtin predicates:

- **empower** is a predicate over domains $S \times R$. If s a subject and r a role, then $empower(s, r)$ means that subject s is empowered in role r .
- **use** is a predicate over domains $O \times V$. If o is an object and v is a view, then $use(o, v)$ means that object o is used in view v .
- **consider** is a predicate over domains $A \times \mathbb{A}$. If α is an action and a is an activity, then $consider(\alpha, a)$ means that action α implements the activity a .
- **hold** is a predicate over domains $S \times A \times O \times C$. If s a subject, α an action, o an object and c a context, $hold(s, \alpha, o, c)$ means that context c holds between subject s , action α and object o .
- **permission, prohibition and obligation** are predicates over domains $R_s \times A_a \times V_o \times C$. Where $R_s = R \cup S$, $A_a = \mathbb{A} \cup A$ and $V_o = V \cup O$. More precisely, if g is a role or a subject, t is a view or an object and p is an activity or an action, then $permission(g, p, t, c)$ (resp. $prohibition(g, p, t, c)$ or $obligation(g, p, t, c)$) means that, grantee g is granted permission (resp. prohibition or obligation) to perform privilege p on target t in context c .

Since in the OrBAC model it is possible to specify both permissions and prohibitions, some conflicts may occur. In prioritized OrBAC [5] the authorization rules are associated with priorities in order to evaluate their significance in conflicting situations. Then, predicates $permission(g, p, t, c)$ and $prohibition(g, p, t, c)$ are replaced by: $permission(g, p, t, c, l)$ and $prohibition(g, p, t, c, l)$, where l is the priority level.

The OrBAC model is self administrated, that is the concepts used to define an administration policy are similar to the ones presented in this section. We give in the following section the basic concepts of the AdOrBAC model.

2.2. AdOrBAC model

The approach in AdOrBAC is to define administration functions by considering different administrative views. Objects belonging to these views have specific semantics; More precisely, we shall consider in the following two administrative views called *role_assignment* and *license* views. They are respectively interpreted as an assignment of a user to a role in the *role_assignment* view and a permission to a role or to a user in the *license* view.

Intuitively, inserting an object in these views will enable an authorized user to respectively assign a user to a role, assign a permission to a role or assign a permission to a user. Conversely, deleting an object from these views will enable a user to perform a revocation.

Defining the administration functions in AdOrBAC then corresponds to specifying which role is permitted to have an access to administrative views. So that only valid licenses can be created.

The AdOrBAC approach is homogeneous with the remainder of the OrBAC model. The syntax used to define permission to administer the policy is completely similar to the remainder of OrBAC.

The two administrative views *license* and *role_assignment* are defined as follows:

License view is used to specify and manage the security policy. Objects belonging to the license view have the following attributes: *grantee*: subject to which the license is granted, *privilege*: action permitted by the license, *target*: object to which the license grants an access and *context*: specific conditions that must be satisfied to use the license.

The existence of a valid license is interpreted as a permission by the following rule:

$permission(Sub, Act, Obj, Context)$:-
 $use(L, license), grantee(L, Sub),$
 $privilege(L, Act), target(L, Obj),$
 $context(L, Context).$

Role assignment view is associated with the following attributes: *assignee*: subject to which the role is assigned and *assignment*: role assigned by the role assignment.

There is the following rule to interpret objects of the *role_assignment* view:

$empower(Subject, Role):-$
 $use(RA, role_assignment),$
 $assignee(RA, Subject), assignment(RA, Role).$

3 Delegation model

We define in this section our delegation model. We show that the expressiveness of AdOrBAC is sufficient to manage delegation requirements without adding new components to the model.

We present in this section the main delegation characteristics [2] like totality, permanence, monotonicity, levels of delegation, multiple delegation, cascading revocation, grant-dependency and show how to model them using the OrBAC formalism.

The approach we suggest to manage delegation is the use of the notion of contexts and administrative views as defined in AdOrBAC.

We define two administrative views: the *license_delegation* view and the *role_delegation* view. These views are used, respectively, to delegate rights (partial delegation) and roles (total delegation) and they are defined as follows:

$permission(GR, A, O, C) :-$
 $use(L, license_delegation), grantee(L, GR),$
 $privilege(L, A), target(L, O), context(L, C).$
 $empower(GR, Role) :-$
 $use(RD, role_delegation),$
 $assignee(RD, GR), assignment(RD, Role).$

Objects belonging to these views have the same attributes, respectively, as the *license* view and the *role_assignment* view but also have an additional attribute called *grantor* : the subject who is creating the license or the role.

Inserting an object in the *license_delegation* view or in the *role_delegation* view will enable a grantor to respectively delegate permission and role to a grantee. Therefore to manage delegation policy we must define which grantor (role or user) have an access to these views and in which context. This is defined by facts having the following form:
 $permission(gr, delegate, license_delegation, context).$
 $permission(gr, delegate, role_delegation, context).$

To illustrate the approach, let us consider a situation where there are two users, *John* a professor and *Mary* his secretary. The role secretary is not generally permitted to

have an access to the view *stud_notes*. However, *John* decides to delegate to *Mary* a permission to update his student's notes. Obviously, *John* must have a permission to delegate this right.

For this purpose, the administrator should first create the following administrative view:

$use(L, note_delegation):-$
 $use(L, license_delegation),$
 $privilege(L, update), target(L, stud_notes).$

The view *note_delegation* is derived from *license_delegation* view and only contains licenses to update student's notes.

The administrator should then give to the role professor the permission to delegate licenses in this view:

$permission(prof, delegate, note_delegation, nominal).$
 where *nominal* represents the default context.

Using this permission, *John* can delegate to *Mary* a permission to update his student's notes by creating a new license L_1 , in the *note_delegation* view, with the following attributes: *grantee*: *Mary*, *privilege*: *update*, *target*: *John_stud_notes* (which is a sub-view of *stud_notes*) and *context*: *nominal*. As a result, the following permission is created:

$permission(mary, update, john_stud_notes, nominal).$

Notice here that *John* can delegate the permission to update the view *John_stud_notes* because this is a sub-view of *stud_notes*.

Therefore we assume that if the grantor have the permission to delegate the license L then he also have the permission to delegate a license L' , which is a sub_license of L .

We shall now formally define different types of delegation parameters, namely permanence, monotonicity, levels of delegation, multiple delegation; cascading revocation and grant dependent revocation. For this purpose, we need define the following predicate to verify if the license L is a sub_license of L' :

$sub_license(L, L'):-$
 $target(L, T), target(L', T'), sub_target(T, T'),$
 $privilege(L, P), privilege(L', P'), sub_priv(P, P'),$
 $context(L, C), context(L', C'), sub_context(C, C').$

We consider O is a sub_target of O' if there are two views and O is a sub-view of O' , or if O is an object used in the view O' , or if they are equal.

$sub_target(O, O'):-$
 $sub_view(O, O'); use(O, O'); O = O'.$

Similarly, we define predicates *sub_privilege* and *sub_context*.

We also define the following predicate to verify if licenses L and L' are equivalent:

$equiv_licenses(L, L'):-$
 $sub_license(L, L'), sub_license(L', L).$

3.1. Permanence

Permanence refers to types of delegation in terms of their time duration. Indeed, in some circumstances, the delegation only applies temporarily and will be automatically revoked after a given deadline. This may be modeled in our approach by simply using a temporal context. For further details about the context definition see [6].

In the previous example, there is no temporal specification of time duration in the context of delegation, so the delegation is permanent. Now if we assume that *John* wants to delegate to *Mary* the permission to update his student's notes only during his vacation, then he must specify this condition in the delegation context associated with the new license L_2 he creates for *Mary*. The license L_2 is similar to L_1 except that $\text{context} = \text{during_John_vacation}$. The delegated permission is specified as follows:

```
permission(mary, update, john_stud_notes,  
           during_john_vacation).
```

3.2. Monotonicity

Monotonic delegation means that upon delegation the grantor maintains the permission he has delegated, as described in the example of previous sections. On the other hand, with a non-monotonic delegation, the grantor loses this permission for the duration of the delegation.

To model non-monotonic delegation we define the *license_transfer* view as follows:

```
use(L, license_delegation):-  
  use(L, license_transfer).  
prohibition(Sub, Act, Obj, C, Max):-  
  use(L, license_transfer),  
  grantor(L, Sub), privilege(L, Act),  
  target(L, Obj), context(L, C).
```

The *license_transfer* view is a sub_view of the *license_delegation* view. So, inserting an object in this view will create a new permission to the grantee. In addition, it will create an interdiction to the grantor associated with the highest priority level Max. Therefore, the grantor will lose the permission he has delegated.

Note that, the context of the prohibition and the delegated permission is the same one. So the grantor will lose this permission only for the time of the delegation.

3.3 Multiple delegation

Multiple delegation refers to the number of grantees to whom a grantor can delegate the same right at any given time. To control the delegation we assume that this number (Nm) is fixed by the administrator using the context

max_multi_delegation. In simple delegation case Nm is equal to 1:

```
permission(subject, delegate, view, context  
           &max_multi_delegation(Nm)).
```

To define the context *max_multi_delegation* we need to count the delegation number concerning the same grantor and the same right:

```
hold(S, A, L, max_multi_delegation(Nm)):-  
  use(L, license_delegation),  
  grantor(L, S), count(L', use(L', license_delegation),  
  grantor(L', S), equiv_licenses(L, L'), Nm'),  
  Nm' <= Nm.
```

We assume that $\text{count}(V, p(V), N)$ is a predicate that count the set of instances of variable V that satisfies predicate $p(V)$. N represents the result of the count predicate.

Note that, we consider the licenses L and L' are the same right since they are equivalent.

To explain this, let us consider the same roles of the previous example. Suppose now we are in a simple delegation case, so that *John* can delegate the right to update his student's notes only for one time:

```
permission(john, delegate, note_delegation,  
           max_multi_delegation(1)).
```

Thus, if *John* delegate to *Mary* the permission to update the view *John_students_notes*, then *John* does not have the permission to delegate this right to another user. For instance, he cannot delegate the permission to update the file *master_stud_notes*, which belongs to the view *John_stud_notes*, to his assistant since this right is a sub_license of the first one.

Notice that when the *max_multi_delegation* context is not used, the number of permissions a subject can delegate is not restricted. So this subject can delegate as many licenses he wants.

3.4 Level of delegation

This characteristic defines whether or not each delegation can be further delegated and how many times.

For this purpose, we define the *grant_option_licence* view as follows :

```
permission(U, delegate, Licence, C&valid_level):-  
  use(L, grant_option_licence), grantee(L, U),  
  target(L, Licence), context(L, C).
```

The context *valid_level* is defined as follows:

```
hold(U, delegate, L, valid_level):-  
  use(L', grant_option_licence), sub_licence(L', L),  
  grantor(L', U), level(L, V), level(L', V'), V' < V.
```

Objects belonging to the *grant_option_licence* view have an additional attribute called *level*: the number of authorized delegation steps.

Inserting a license in this view will create a permission to the grantee to delegate the right but only in the context `valid_level`.

This means that, if we consider the same license L_1 of the previous example and suppose that *John* wants to grant his secretary the permission to delegate this license with a delegation level equal to 3.

For this purpose, *John* creates in the `grant_option_licence` view the licence L_3 with the following attributes: `grantee`: *Mary*, `privilege`: `delegate`, `target`: L_1 , `level`: 3, `context`: `nominal`.

This corresponds to the following rule:

`permission(mary, delegate, L1, valid_level).`

Therefore *Mary* can delegate the license L_1 (or a sub.license of L_1) in the context `valid_level`, which means that she can create a license L_4 to grant another user to delegate this license and the delegation level of L_4 must be lower than 3, since the delegation level of L_3 is equal to 3.

The grantor can also restrict the scope of the delegation using conjunctive context. For instance, *John* can specify, in the delegation context, that *Mary* can grant another user to delegate L_1 only during her vacation:

`permission(mary, delegate, L1, valid_level
&during_mary_vacation).`

In this case, *Mary* can further delegate the license she receives from *John* but the delegated license will only apply in the context `during_Mary_vacation`.

Note that we consider here only the monotonic and the partial delegation. The `grant_option_licence` view is also used in the total delegation case (multi-step role delegation) and non_monotonic delegation case (multi-step transfer). A more detailed model will be proposed in future work.

3.5 Revocation

Revocation is an important aspect in delegation models. In this section we present some revocation properties and we plan to give a more detailed presentation in a forthcoming paper.

Grant Dependency In the case of Grant_Dependent revocation (GD) only the grantor is allowed to revoke the delegated license or role. On the other hand, Grant_Independent revocation (GID) allows any member in the sponsoring role to revoke the grantee. This is modeled using contexts:

`permission(subject, revoke, license_delegation, gd).`
`permission(subject, revoke, license_delegation, gid).`

The `gd` and `gid` contexts are defined as follows:

`hold(User, revoke, L, gd):-
use(L, license_delegation), grantor(L, User).`
`hold(User, revoke, L, gid):-`

`use(L, license_delegation), grantor(L, GR),
empower(GR, Role), empower(User, Role).`

Contexts `gd` and `gid` are relevant for license revocation, we can similarly define `gdr` and `gidr` to revoke a role.

Cascading revocation In multi-step delegation it is necessary to give the possibility to revoke indirectly the delegation chain.

We can model this property thanks to the contextual license: the delegation of right is valid only if the grantor still has this right.

For this purpose we define the view `cascading_delegation`, which is a sub_view of `license_delegation` view, as follows:

`permission(Sub, Act, Obj, C&valid_deleg(gr)):-
use(L, cascading_delegation), grantee(L, Sub),
grantor(L, gr), privilege(L, Act),
target(L, Obj), context(L, C).`

Inserting an object in this view will create a permission with an additional context (`valid_deleg context`) which verify if the grantor still has his right.

This context is defined as follows:

`hold(User, A, O, valid_deleg(gr)):-
is_permitted(User, A, O).`

Therefore, the delegated permission is valid only if the delegation chain is maintained.

Note that this property only concerns monotonic delegation. Other revocation aspects remain to be investigated in further work.

4 Decidability and complexity

The OrBAC model is based on first order logic and more precisely on Datalog [11] which ensures a decidable and tractable theory.

Datalog programs do not allow the use of functional terms and must only include both defined and safe rules. A rule is defined if every variable that appears in the conclusion also appears in the premise. A rule is safe if it only provides means to derive a finite set of new facts. In pure Datalog program, rules do not contain any negative literal. Pure Datalog guarantees that any access control policy will be decidable in polynomial time. However pure Datalog expressivity is very restricted.

In Datalog[¬], the negation restriction is relaxed. Negative literals are allowed but rules must be stratified [11]. A stratified Datalog[¬] program is computable in polynomial time.

The definition of security policies using the OrBAC model obeys the Datalog[¬] restriction except the definition of contexts through the `hold` predicate. More precisely, the

security rules correspond to ground close facts specified using the *permission*, *prohibition* predicates. Specifications of predicates *empower*, *use* and *consider* correspond also to facts or rules that respect the Datalog⁻ restriction.

By contrast, the definition of contexts does not correspond to Datalog⁻ restriction for the following reasons: these rules contain functional terms and are not always safe and defined.

To solve these problems it is proposed firstly, to restrict the theory so that only *relevant* contexts are evaluated. A context is relevant if it appears in the definition of a security rule. Secondly, a relevant context is always fully instantiated. Finally it is proposed to pre-compute the evaluation of the *Empower*, *Use* and *Consider* predicates using a bottom-up strategy. Then, the evaluation of queries is completed using the top-down strategy as defined in the SGL algorithm [10]. This hybrid strategy guarantees the decidability of query evaluation in the OrBAC model and its termination in polynomial time.

5 Related work

The previous work on delegation has shown that delegation is a complex concept and, to our best knowledge, there is no complete model for describing all delegation characteristics such as multiple delegation, cascading revocation, etc.

Proposed models are based on RBAC which is not expressive enough to deal with the delegation requirements. To solve this problem it is suggested to extend the RBAC model to include delegation components, such as new types of roles, actions, permissions, etc. Unfortunately, this is a complex task to manage, since it is necessary to add new components for modeling every delegation characteristic.

For instance, in the RBDM0 model proposed by [2], authors extend RBAC₀ model to define role-based delegation. They define a relation *can-delegate* $\subseteq R \times R$ to control role delegation and add new components such as: *Users-O* and *Users-D* to differentiate between original and delegated members, *UAO* and *UAD* to specify original member assignment and delegate member assignment relations, etc.

They also propose some extensions to RBDM0 to address more delegations characteristics. This requires additional components. For instance they add new types of permissions: *delegable* and *non delegable* permissions (*permissions-PN* and *permissions-PD*) to model partial delegation.

In RBDM1 [3], an extension of RBDM0, is proposed. This model adds new components such as a partially ordered role hierarchy relation $RH \subseteq R \times R$ to model delegation using hierarchical roles.

The PBDM model [13] is another delegation model based on RBAC96. This model uses the *can-delegate* re-

lation with prerequisite condition to restrict delegates, and adds new types of roles and permissions to address permission level delegation requirement. In PBDM0 roles are partitioned into regular roles (RR) and delegation roles (DTR). This partition induces a parallel partition of the two RBAC components: user-role assignment (UA) and permission-role assignment (PA). UA is separated into user-regular role assignment (UAR) and user-delegation role assignment (UAD). PA is similarly separated into permission-regular role assignment (PAR) and permission-delegation role assignment (PAD).

PBDM1 is an extension to PBDM0 which supports security administrator involved delegation and revocation. This model adds new components such as delegatable roles (DBR), user-delegatable role assignment (UAB) and permission-delegatable role assignment (PAB).

PBDM2 model is another extension which addresses a role-to-role delegation. Like other models, PBDM2 adds new components, such as temporal delegatable roles (TDBR), and redefines existing ones.

The ABDM model [12] is an attribute-based delegation model, which extend PBDM model to address delegation constraint. This model redefines the *can-delegate* relation to restrict the delegation. The ABDM model is also extended to ABDM_x for more flexibility.

Another delegation model is proposed in [4]. This model is more complete than previous works. But it also extends RBAC96 by adding new components to specify more delegation characteristics like temporary non-monotonic delegation. It introduces the relations: *can-delegate* and *can-receive* to authorize role delegation, and the relations *can-delegatep* and *can-receivep* to authorize permission delegation. Also, it defines actions like *xferR₀*, *xferP₀*, *xferP₁*, etc, to model roles and permissions transfer. The two relations *tempUA* and *tempPA* are introduced to record temporary user-role and user-permission delegations.

Like the above discussed works, this model introduces new relations or actions to model each delegation characteristics. This is a complex task to manage especially when the delegation model has to be enriched.

Compared to these works, our model is more flexible, simpler to manage and more complete. Indeed, OrBAC model offers facilities to deal with delegation requirements without the need for additional components.

Namely, OrBAC model is based on multi-granular and contextual licenses. This provides facilities to define many delegation characteristics like totality, permanence, revocation, etc.

Moreover, thanks to the use of views we can express a large number of conditions, which allow us to specify delegation constraints; This is modeled by prerequisite conditions associated with the grantee or the grantor. For instance, the professor is permitted to delegate a permission

to manage her courses to her assistant but only if this assistant is a graduate student.

6 Conclusion

In this paper, we have proposed a new delegation approach for role-based access control. We have showed that it is possible to specify delegation requirements using the OrBAC formalism. This model is self administrated and offers facilities, such as multi- granular license, contextual license, use of views, etc., which gives means to specify delegation characteristics without adding new components or modifying the exiting ones. Therefore our approach is more flexible, simpler and more complete than previous works based on RBAC model.

The future work will be dedicated to enrich our delegation model and more precisely the revocation mechanism. We intend to include several of the revocation schemes as described in [8].

Acknowledgment

This work is partially supported by the RNRT project Politess. For this work, Meriam Ben Ghorbel-Talbi is funded by the IFC (Institut Francais de Cooperation en Tunisie).

References

- [1] A. Abou-El-Kalam, S. Benferhat, A. Miège, R. E. Baida, F. Cuppens, C. Saurel, P. Balbiani, Y. Deswarte, and G. Trouessin. Organization Based Access Control. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '03)*. IEEE Computer Society, June 2003.
- [2] E. Barka and R. Sandhu. A Role-based Delegation Model and Some Extensions. In *Proceedings of the 23rd National Information Systems Security Conference (NISSC'00)*, Baltimore, MD, October 2000.
- [3] E. Barka and R. Sandhu. Role-Based Delegation Model/ Hierarchical Roles (RBDM1). In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, Tucson, Arizona, December 2004.
- [4] J. Crampton and H. Khambhammettu. Delegation in Role-Based Access Control. In *Proceedings of the 11th European Symposium on Research in Computer Security (ESORICS'06)*, Hamburg, Germany. Springer, September 2006.
- [5] F. Cuppens, N. Cuppens-Bouahia, and M. Ben-Ghorbel. High Level Conflict Management Strategies in Advanced Access Control Models. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 186:3–26, July 2007.
- [6] F. Cuppens and A. Miège. Modelling Contexts in the OrBAC Model. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC '03)*, Las Vegas, Nevada. IEEE Computer Society, December 2003.
- [7] F. Cuppens and A. Miège. Administration Model for OrBAC. *International Journal of Computer Systems Science and Engineering (CSSE)*, 19(3), May 2004.
- [8] A. Hagstrom, S. Jajodia, F. Parisi-Persicce, and D. Wijesekera. Revocation - a Classification. In *Proceedings of the 14th Computer Security Foundation Workshop (CSFW'01)*, Cape Breton, Nova Scotia, Canada. IEEE Computer Society, June 2001.
- [9] R. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.
- [10] D. Toman. Memoing Evaluation for Constraint Extensions of Datalog. *Constraints*, 2(3/4):337–359, 1997.
- [11] J. D. Ullman. *Principles of Database and Knowledge-Base Systems: Volume II: The New Technologies*. W. H. Freeman & Co., New York, NY, USA, 1990.
- [12] C. Ye, Z. Wu, and Y. Fu. An Attribute-Based Delegation Model and Its Extension. *Journal of Research and Practice in Information Technology*, 38(1), 2006.
- [13] X. Zhang, S. Oh, and R. Sandhu. Pbdm: A Flexible Delegation Model in RBAC. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT'03)*, Como, Italy. ACM Press, June 2003.