Policy Administration in Tag-Based Authorization

Sandro Etalle[1,2], Timothy L. Hinrichs[3], Adam J. Lee[4], Daniel Trivellato[1], and Nicola Zannone[1]

[1] Eindhoven University of Technology
[2] University of Twente
[3] University of Illinois at Chicago
[4] University of Pittsburgh

Abstract. Tag-Based Authorization (TBA) is a hybrid access control model that combines the ease of use of extensional access control models with the expressivity of logic-based formalisms. The main limitation of TBA is that it lacks support for policy administration. More precisely, it does not allow policy-writers to specify administrative policies that constrain the tags that users can assign, and to verify the compliance of assigned tags with these policies. In this paper we introduce TBA^2 (Tag-Based Authorization & Administration), an extension of TBA that enables policy administration in distributed systems. We show that TBA^2 is more expressive than TBA and than two reference administrative models proposed in the literature, namely HRU and ARBAC97.

Keywords: access control, policy administration, auditing.

1 Introduction

Access control systems in real-world organizations are mostly based on extensional approaches to access control (e.g., access control lists), as their ease of use is preferred to the flexibility of logic-based models. Authorization policies in extensional models are based on simple assignments of rights to users, or on the characterization of users in terms of properties (e.g., roles) and the assignment of rights based on those properties. Nevertheless, the lack of expressiveness of extensional models severely limits the constraints that can be expressed in authorization policies. A hybrid approach to access control that combines the usability of extensional models and the flexibility and expressiveness of logic-based formalisms would offer great benefits for the deployment of access control systems in real-world organizations.

To accommodate this need, Tag-Based Authorization (TBA) has been proposed by Hinrichs et al. [15], based on the work by Najafian Razavi and Iverson [20] and Wang et al. [26]. TBA is a hybrid access control model that relies on formal logic for the definition of authorization policies, and on extensional models for describing a system's subjects and objects in terms of simple properties (e.g., roles). This integration allows relatively untrained users to choose descriptive tags for the system's subjects and objects; security experts then write logical policies that define access authorizations using combinations of those tags. The resulting access control model is flexible and easy to use, yet expressive enough to match the needs of complex application domains.

As an example application domain for the TBA model, consider Operation Atalanta, a military operation involving several EU navies that collaborate to prevent criminal activities (such as smuggling and pirate attacks) off the Somali coast. The information gathered and exchanged by the collaborating vessels is typically processed and classified by the vessels' operators (e.g., a transiting ship is marked as suspicious). Yet, the policies for accessing this information are regulated by complex context- and content-based conditions and must be written by higher-rank personnel. By explicitly distinguishing the task of characterizing information and the task of writing the authorization policy, TBA leads to a better separation of duties and valorization of the skills of a vessel's workforce than the existing access control models.

If on the one hand allowing low-rank users to assign tags to subjects and objects greatly enhances the usability of the access control system, on the other hand it enables low-rank users to influence the system's authorizations. In fact, inaccurate tags (whether by intention or accident) can circumvent the intended authorization policy of the system. The main limitation of TBA in this respect is the lack of support for policy administration. In particular, TBA does not allow policy-writers to: (1) define which users may assign which tags to which subjects and objects, i.e., to specify *administrative policies*; (2) hold users accountable for the tags they assign, and verify the compliance of tags with administrative policies; and (3) revoke incorrect tags or tags assigned by unauthorized users. In the scenario above, for instance, it is important to allow only operators with appropriate clearance to tag sensitive information, and to verify whether the assigned tags comply with this policy (possibly revoking the tags that do not satisfy the policy).

In a centralized system, these issues can be addressed by means of a traditional access control mechanism that regulates the tagging process and a logging mechanism that records users' actions. In a distributed system, however, these solutions are insufficient as they would require entities in one security domain to trust the enforcement and logging mechanisms of systems in different security domains. Furthermore, tracing the objects exchanged in a distributed system might be difficult, complicating or even preventing the revocation process.

In this paper, we extend the TBA model from [15] to enable policy administration in distributed systems. In particular, we introduce TBA^2 (Tag-Based Authorization & Administration), an extension of TBA that allows security administrators to specify constraints on the tagging process and to verify the compliance of tags with administrative policies by combining traditional "a priori" access control mechanisms with "a posteriori" verification. Technically, our extension consists of associating an *issuer* to each tag, which identifies the user who assigned the tag. For example, all the tags issued by an operator will be marked with the identifier of the operator. In contrast to previous work on a posteriori verification (e.g., [7]), this enables policy compliance verification without the need of an auditing infrastructure, by exploiting the observability of users' actions (i.e., "signed" tags). As a consequence, TBA^2 represents a lightweight solution for the enforcement of authorization and administrative policies in distributed systems. We show that the proposed model is more expressive than the original TBA model and than two well-known administrative models proposed in the literature, namely the Harrison, Ruzzo, and Ullman model [13] and ARBAC97 [23].

The paper is organized as follows. Section 2 reviews the TBA model and discusses its limitations. Section 3 introduces TBA^2, and Section 4 evaluates its expressive power. Section 5 discusses related work, and Section 6 concludes the paper.

2 Background

Tag-Based Authorization (TBA) [15] is a hybrid access control model combining the flexibility and expressiveness of logic-based formalism with the ease of use of extensional models. In this section we first present the definitions of the TBA model given in [15] that are relevant for this paper, and then we identify the main limitations of TBA with respect to its deployment in a distributed system.

2.1 The TBA Model

In this paper we use S to denote the set of subjects, O to denote the set of objects, and R to denote the set of rights that can be assigned within a system. T denotes the set of possible tags that can be assigned to S and O, and $\mathsf{tag} : S \cup O \rightarrow 2^T$ is a function that maps a subject or object to the set of tags assigned to it. Tag denotes the set of all possible tag functions. A TBA authorization policy is written in some logical access control language $\langle P, L, \models \rangle$ defined as follows.

Definition 1 (TBA). *For a function* tag *and a logical language* $\langle P, L, \models \rangle$*, where*

- tag*:* $S \cup O \rightarrow 2^T$ *maps a subject or object to the set of tags assigned to it*
- P*: the set of all authorization rules*
- L*: the set of queries including* allow(*s,o,r*) *for all subjects* s*, objects* o*, rights* r
- $\models$*: a subset of* $P \times \mathsf{Tag} \times L$

$\mathsf{auth}_{TBA}(s, o, r)$ *if and only if* $P', \mathsf{tag} \models \mathsf{allow}(s, o, r)$ *for some* $P' \subseteq P$*.*

The operator $\models$ dictates which queries are true given the set of authorization rules P and a function tag. The following example based on the scenario introduced in Section 1 illustrates TBA using Datalog as the policy language.

Example 1 (TBA). Consider two subjects s_1 and s_2 and two objects o_1 and o_2 that are tagged as follows:

- tag(s_1) = {uk_navy,operation_atalanta,operations_specialist}
- tag(s_2) = {fr_navy}
- tag(o_1) = {cargo_ship,radar}
- tag(o_2) = {gulf_of_aden,sat_732,high_res}

Further, consider the following policy rules:

1. allow(S_x,O_x,read) :- uk_navy $\in$ tag(S_x), cargo_ship $\in$ tag(O_x)
2. allow(S_x,O_x,read) :- fr_navy $\in$ tag(S_x), cargo_ship $\in$ tag(O_x)
3. allow(S_x,O_x,read) :- operations_specialist $\in$ tag(S_x), radar $\in$ tag(O_x)
4. allow(S_x,O_x,read) :- uk_navy $\in$ tag(S_x), operation_atalanta $\in$ tag(S_x), high_res $\in$ tag(O_x), sat_732 $\in$ tag(O_x)

This policy allows the members of the British and French Navy (denoted uk_navy and fr_navy respectively) to access documents about cargo ships (rules 1 and 2), operations specialists to access documents about radar systems (rule 3), and all members of the British Navy serving on Operation Atalanta to access high resolution satellite photographs taken by sat_732 (rule 4). As a result, subject s_1 can access objects o_1 and o_2, while subject s_2 can only access object o_1.

Tag-based authorization differs from standard logical access control models in that the tag function has a fixed semantics defined outside of the policy. In particular, the semantics of tag is defined by the users of a system, who assign tags to the system's subjects and objects. The fixed semantics of tag forces security administrators to define authorization policies at a higher level of abstraction than the usual $S \times O \times R$. More precisely, TBA policies are defined over the space of tags $2^T \times 2^T \times R$, i.e., subjects and objects are replaced by tag sets. This abstraction might result in a less flexible system if the tag-space is not exhaustive enough to accommodate a particular situation; however, the model can be easily adapted to define policies over $(S \cup 2^T) \times (O \cup 2^T) \times R$ (which combines the space of tags and the ones of subjects and objects) by adding to the set of tags T a tag identifying each subject and object in a system.

In [15], the authors evaluate the expressive power of TBA by expressing a range of well-known policy idioms. In particular, they show that TBA can be successfully employed to represent an access matrix, attribute-based access control policies, role-based access control policies, discretionary access control, mandatory access control, and three rule types of the RT [18] policy language (namely, all rule types except for "linked roles"). In addition, their results show that TBA is strictly more expressive than common access control models such as SDCO [21], ARBAC97 [23], and BLP [3].

2.2 Limitations of TBA

The applicability of TBA is due to a key observation: within a system, the users responsible for creating and categorizing (i.e., tagging) data are usually not in charge and do not have the expertise to define the security policies governing the system. For example, on a navy vessel some operators have the task of analyzing the surrounding maritime traffic and identifying suspicious behaviors, other operators gather intelligence and add details to these suspicious activities, etc. The policy governing the access to this information, on the other hand, is defined by the authorities in command of operations.

By assigning tags to subjects and objects, however, users directly influence the authorizations within a system. For this reason, it is necessary to enable policy-writers (e.g., security administrators) to control the tagging process. In this respect, we identify two key issues that are not addressed by the TBA model:

1. *Administrative policies:* Security administrators should be able to specify policies defining which users are allowed to assign and revoke which tags to which subjects and objects. For instance, the security administrator of a navy vessel should be able to restrict to the commanding officer the right to promote the vessel's officers to higher ranks. In addition, it should be possible to regulate the propagation of users' rights, i.e., the extent to which a user can delegate its tagging rights to other users.

2. *Tag Verification and Revocation:* It should be possible to hold users accountable for the tags they assign, and to verify the compliance of tags with administrative policies by checking who assigned them. Consider, for instance, a document containing information about a suspected pirate attack, distributed by the French Navy to its allies in Operation Atalanta. If some tag is added to the document by a user outside the navy's system, the security administrator of the French Navy should be able to verify whether the user was authorized to label the document (e.g., if the user is an operator of an allied navy, rather than an unknown subject). Accordingly, "invalid" tags identified in the verification process should be revoked.

Even though some simple administrative policies could be expressed in TBA, the fact that the "issuer" of a tag is not taken into account by the model makes it impossible to specify constraints that link tags based on the subject who assigned them (e.g., RT's linked roles [18]). Therefore, TBA does not allow the specification of rules such as "a subject can revoke only the tags that she assigned", or "a subject can mark a document as sensitive only if she is (tagged as) a senior officer by a navy that the EU labels as member of Operation Atalanta".

More than the specification of administrative policies, however, the major limitation of TBA is represented by the lack of mechanisms for verifying their enforcement. In a centralized system, administrative polices can be enforced by means of an access control system that governs the tagging process. Tag verification can be achieved by means of a logging mechanism that records all the tags assigned by the system users. Security administrators can then simply audit these logs to verify their compliance with the system's policies, and revoke the invalid tags identified in the process. In a distributed system, however, these solutions are insufficient for the following reasons:

1. They require entities in one security domain to trust the administrative policies (and their enforcement) of systems in different security domains.
2. Since tags may be assigned by users of different systems, and the issuer of a tag is not considered by the TBA model, verifying the compliance of the tags assigned by a subject with respect to administrative policies might not be feasible. In fact, this might require inspecting the logs of possibly all the systems in the distributed system. It is unlikely, however, that a system would disclose its logs to systems from a different security domain for auditing purposes. In addition, as information is exchanged between different systems, tracing the tags assigned by a certain user (e.g., to revoke them) becomes very difficult, if not impossible.

In the next section we show how the TBA model can be extended to address these limitations.

3 The TBA^2 Model

In this section we present the Tag-Based Authorization & Administration (TBA^2) model, an extension of TBA that enables policy administration in distributed systems. TBA^2 extends TBA by associating an *issuer* to each tag, which identifies the user who assigned the tag. Intuitively, tags become signed statements issued by a user within a system. Formally, we modify the definition of the `tag` function introduced in Section 2.1 as follows:

$\mathsf{tag} : S \cup O \rightarrow 2^{S \times T}$, which returns the set of issuer-tag pairs associated to a subject or object. Representing tags as signed statements is a first step towards addressing the limitations of TBA mentioned in Section 2.2. In the next subsections we discuss how TBA^2 can solve those limitations in details.

For the sake of simplicity, the solution we propose is based on the assumption that the set T of possible tags that a user can assign is common to all the systems in the distributed system. In a real-world distributed system, however, the set of assignable tags might vary from system to system, as entities in different security domains might employ different terms to denote similar concepts. Semantic alignment techniques [14, 24] could be required to align the systems' vocabularies. An additional assumption we make is that each subject and object *belongs* to one system, which represents the security domain where a subject operates, or the system that owns an object (or that has exclusive rights on it). Given a subject or object identifier, it is possible to determine the system to which the subject or object belongs.

3.1 Authorization & Administration Policies

Administrative policies constrain the tags that a user is authorized to assign or revoke. The advantage of TBA^2 with respect to TBA is that it links every tag to the user who assigned it, enabling security administrators to specify fine-grained administrative policies.

TBA^2 requires the set of rights R to include the rights $assign\_tag$ and $revoke\_tag$. Then, TBA^2 is defined as follows.

Definition 2 (TBA^2). *For a function* tag *and a logical language* $\langle P \cup A, L, \models \rangle$*, where*

- tag*:* $S \cup O \rightarrow 2^{S \times T}$ *returns the issuer-tag pairs associated to a subject or object*
- P*: the set of all authorization rules*
- A*: the set of all administrative rules*
- L*: the set of queries including* $\mathsf{allow}(s, o, r)$*,* $\mathsf{allow}(s, o, r', ST)$*, and* $\mathsf{allow}(s, s', r', ST)$ *for all subjects* s *and* s'*, objects* o*, rights* r*, right* $r' \in \{assign\_tag, revoke\_tag\}$*, and set of signed tags* $ST \subseteq 2^{S \times T}$
- $\models$*: a subset of* $(P \cup A) \times \mathsf{Tag} \times L$

$\mathsf{auth}_{TBA^2}(s, o, r)$ *if and only if* $P', \mathsf{tag} \models \mathsf{allow}(s, o, r)$
$\mathsf{auth}_{TBA^2}(s, o, r', ST)$ *if and only if* $A', \mathsf{tag} \models \mathsf{allow}(s, o, r', ST)$
$\mathsf{auth}_{TBA^2}(s, s', r', ST)$ *if and only if* $A', \mathsf{tag} \models \mathsf{allow}(s, s', r', ST)$

for some $P' \subseteq P$*,* $A' \subseteq A$*.*

The following example presents TBA^2 authorization and administrative policies.

Example 2 (TBA^2). Consider three subjects s_1, s_2, and s_3 and an object o belonging to a system governed by the British Navy, which are tagged as follows:

- $\mathsf{tag}(s_1)$ = {(uk_navy,senior_officer)}
- $\mathsf{tag}(s_2)$ = {(uk_navy,junior_officer)}
- $\mathsf{tag}(s_3)$ = {(uk_navy,junior_officer)}
- $\mathsf{tag}(o)$ = {(uk_navy,secret)}

The subjects' tags are issued by the British Navy (denoted as "uk_navy") and indicate that subject s_1 has rank senior_officer, while s_2 and s_3 have rank junior_officer. Object o is tagged by the British Navy as secret. The access to object o and the administration of rights within the system are regulated by the following rules:

1. allow(S_x,o,read) :- (eu,navy) $\in$ tag(S_y), (S_y,senior_officer) $\in$ tag(S_x)
2. allow(S_x,O_x,$assign\_tag$,{(S_x,T_x)}) :- (eu,navy) $\in$ tag(S_y),
   (S_y,senior_officer) $\in$ tag(S_x),
   (eu,navy) $\in$ tag(S_z), (S_z,secret) $\in$ tag(O_x)
3. allow(S_x,S_y,$assign\_tag$,{(S_x,senior_officer)}) :- (eu,navy) $\in$ tag(S_z),
   (S_z,senior_officer) $\in$ tag(S_x),
   (S_z,junior_officer) $\in$ tag(S_y)
4. allow(S_x,O_x,$revoke\_tag$,{(S_x,T_x)}) :- (S_x,T_x) $\in$ tag(O_x)
5. allow(S_x,S_y,$revoke\_tag$,{(S_x,T_x)}) :- (S_x,T_x) $\in$ tag(S_y)

The first rule is an authorization rule stating that object o can be read by a subject S_x if S_x is labeled as senior officer by an EU navy S_y. Rules 2, 3, 4 and 5 are administrative rules. Rule 2 allows senior officers of EU navies to assign tags to objects labeled as secret by any EU navy. Rule 3 allows senior officers of EU navies to assign a *senior_officer* tag to junior officers of the same navy (therefore delegating their rights). Finally, rules 4 and 5 allow the issuer of a tag to revoke the tag. Assuming tag (eu,navy) $\in$ tag(uk_navy), the policy allows subject s_1 to read object o and to assign a *senior_officer* tag to subjects s_2 and s_3.

3.2 Semantics of Administrative Policies

The effects of the exercise by a subject s of the administrative rights $assign\_tag$ and $revoke\_tag$ for a set of (signed) tags ST are shown in Figure 1. The effects of invoking allow(s, o, $assign\_tag$, ST) are intuitive, and imply that a tag st (for each $st \in ST$) is added to the set tag(o). On the other hand, the assignment of a set of tags ST by a subject s to a subject s' can be seen as the delegation of some of the rights (or roles) of s to s'. Delegation can be implemented according to two models: *grant* or *transfer* [8]. In the grant model, after a successful delegation both s and s' are able to benefit from the delegated rights or roles. On the contrary, according to the transfer model subject s loses the delegated rights or roles. Figure 1(a) shows the effects of invoking allow(s, s', $assign\_tag$, ST) using the grant model. The transfer model would imply that all the tags in ST are removed from the set tag(s) after being added to tag(s').

Similarly to the tag assignment operation, also the effects of invoking allow(s, s', $revoke\_tag$, ST) depend on the revocation model employed by the system. To motivate the existence of different revocation models, we describe a scenario based on the tags and rules in Example 2. Assume that senior officer s_1 wants to temporarily delegate her rights to junior officer s_2 because of an emergency. Then, s_1 assigns a *senior_officer* tag to s_2. Later, subject s_2 delegates her rights to s_3, and accordingly assigns tag *senior_officer* to s_3. When the emergency is over, s_1 returns to her regular duties and revokes the *senior_officer* tag from s_2. Now, the question is whether s_3's *senior_officer* tag should also be automatically revoked or not.

allow(s,o,assign_tag,ST)
$\forall\ st \in ST$: $\mathsf{tag}(o) = \mathsf{tag}(o) \cup \{st\}$

(a) Semantics of $\mathsf{allow}(s, o, assign\_tag, ST)$ and $\mathsf{allow}(s, s', assign\_tag, ST)$

allow(s,s′,assign_tag,ST)
$\forall\ st \in ST$: $\mathsf{tag}(s') = \mathsf{tag}(s') \cup \{st\}$

allow(s,s′,revoke_tag,ST)

$\forall\ st \in ST$: $\mathsf{tag}(s') = \mathsf{tag}(s') \backslash \{st\}$
 	
let $ST = \{(s_1, t_1), \ldots, (s_n, t_n)\}$
$\forall\ so \in S \cup O, t \in T$ such that $(s', t) \in \mathsf{tag}(so)$
if $\exists A' \subseteq A$ such that
$A', \mathsf{tag} \models \mathsf{allow}(s', so, assign\_tag, \{(s', t)\})$ if $(s_1, t_1), \ldots, (s_n, t_n) \in \mathsf{tag}(s')$
and $\forall A'' \subseteq A$ we have that
$A'', \mathsf{tag} \not\models \mathsf{allow}(s', so, assign\_tag, \{(s', t)\})$ if $(s_1, t_1), \ldots, (s_n, t_n) \notin \mathsf{tag}(s')$
then
if $so \in S$ then
let ST' be the set of such tags (s', t)
invoke $\mathsf{allow}(s', so, revoke\_tag, ST')$
else
$\mathsf{tag}(so) = \mathsf{tag}(so) \backslash \{(s', t)\}$

(b) Semantics of $\mathsf{allow}(s, s', revoke\_tag, ST)$ with Cascading Revocation

allow(s,s′,revoke_tag,ST)
$\forall\ st \in ST$: $\mathsf{tag}(s') = \mathsf{tag}(s') \backslash \{st\}$ let $ST = \{(s_1, t_1), \ldots, (s_n, t_n)\}$ $\forall\ so \in S \cup O, t \in T$ such that $(s', t) \in \mathsf{tag}(so)$ if $\exists A' \subseteq A$ such that $A', \mathsf{tag} \models \mathsf{allow}(s', so, assign\_tag, \{(s', t)\})$ if $(s_1, t_1), \ldots, (s_n, t_n) \in \mathsf{tag}(s')$ and $\forall A'' \subseteq A$ we have that $A'', \mathsf{tag} \not\models \mathsf{allow}(s', so, assign\_tag, \{(s', t)\})$ if $(s_1, t_1), \ldots, (s_n, t_n) \notin \mathsf{tag}(s')$ then $\mathsf{tag}(so) = \mathsf{tag}(so) \backslash \{(s', t)\} \cup \{(s, t)\}$

(c) Semantics of $\mathsf{allow}(s, s', revoke\_tag, ST)$ with Non-Cascade Revocation

allow(s,o,revoke_tag,ST)
$\forall\ st \in ST$: $\mathsf{tag}(o) = \mathsf{tag}(o) \backslash \{st\}$

(d) Semantics of $\mathsf{allow}(s, o, revoke\_tag, ST)$

Fig. 1. Effects of the invocation of rights $assign\_tag$ and $revoke\_tag$

Two main revocation models have been proposed in the literature. The first model, called *cascading* revocation [4, 12], aims to overturn all the changes to a system authorizations made exploiting the tags being revoked. This implies that if a subject s revokes a set of tags ST from a subject s', then all tags subsequently assigned by s' (and by the subjects to which s' assigned a tag) without other supporting authorizations must be recursively revoked. The effects of invoking $\mathsf{allow}(s, s', revoke\_tag, ST)$ with cascading revocation are shown in Figure 1(b). A domain that typically resorts to cascading revocation is data protection. Whenever an individual revokes the consent (i.e., the right) to process her personal data to a service provider, all the authorizations on the data of the

service provider and of the subcontractors to whom the service provider delegated the processing of the data are revoked.

The dual model of cascading revocation is called *non-cascade* revocation [6] (or *simple* revocation in [4]). In non-cascade revocation, if a subject s revokes a set of tags ST from a subject s', instead of revoking the tags that s' assigned exploiting the authorizations deriving from ST (as done by cascading revocation), these tags are modified as if they were issued by s. Intuitively, this requires s to be allowed to both revoke tags ST from subject s' and to assign tags ST in her place. The rationale behind non-cascade revocation is clarified by the following example. In most organizations, the authorizations that users possess are related to their role within the organization. Suppose there is a change in the role of a user s'. This may imply a change also in the privileges of s': new rights will be granted to s' and some of her previous rights will be revoked. Applying cascading revocation would result in the undesirable effect of deleting all the authorizations that s' granted and, recursively, all the authorizations granted through them, which then might need to be re-issued. Moreover, all the tags assigned by s' that depend on the revoked rights would be invalidated. A better solution to this scenario is to preserve the authorizations granted by user s', possibly substituting s' with another user as the grantor (i.e., issuer) of those authorizations. In [6], for instance, s' is replaced by the user s who is revoking her rights. The semantics of $\mathsf{allow}(s, s', revoke\_tag, ST)$ with non-cascade revocation is shown in Figure 1(c).

According to both semantics presented above, since objects cannot further delegate their rights to other subjects or objects, the effects of invoking $\mathsf{allow}(s, o, revoke\_tag, ST)$ are the same in cascading and non-cascade revocation (Figure 1(d)).

3.3 Tag Verification

Tag verification is the process of verifying the compliance of tags with administrative policies. More precisely, the goal of tag verification is to determine whether a user is (or was) authorized to assign a given tag. Typically, the enforcement of authorization and administrative policies within a system is achieved by means of a priori access control mechanisms. In a distributed system, however, relying exclusively on a priori mechanisms requires entities in one security domain to trust systems in different security domains for policy enforcement. Thanks to the observability of users' actions deriving from the signing of tags, TBA^2 allows security administrators to complement a priori mechanisms with a posteriori tag verification using a lightweight auditing mechanism.

Technically, we say that a tag t assigned by a subject s to an object o (respectively to a subject s') is *valid* if for a set of administrative rules A' and a set of signed tags ST such that $(s, t) \in ST$ we have that

$A', \mathsf{tag} \models \mathsf{allow}(s, o, assign\_tag, ST)$ (resp. $A', \mathsf{tag} \models \mathsf{allow}(s, s', assign\_tag, ST)$)

Otherwise, we say that the tag is *invalid*. This validity check can be used both as an a priori mechanism for the enforcement of administrative policies and a posteriori for auditing purposes.

The verification of a tag (s, t) against the administrative policy of a system might require the verification of a set of tags $(s_1, t_1), \ldots, (s_n, t_n)$ against the policies of systems in different security domains. In fact, to verify whether tag (s, t) is valid, we need

in turn to verify the validity of the *supporting tags* of (s, t), i.e., the tags that authorized subject s to issue (s, t). Consider, for instance, rule 3 in Example 2, which states that "senior officers of EU navies may assign a $senior\_officer$ tag to junior officers of the same navy". To verify whether the $senior\_officer$ tag assigned by subject s_1 to subject s_2 is valid, we need first to confirm that s_1 is actually a senior officer and s_2 has a $junior\_officer$ tag issued by the same navy. This verification process is similar to the credential chain discovery problem in trust management. Accordingly, trust management algorithms [19, 25] can be employed to support the verification of tags' validity.

An additional problem of tag verification is that in a distributed system entities in one security domain might not trust systems in different security domain to perform the validity check. In this respect, we identify three types of trust relationships that can be of interest for tag verification in TBA[2], resulting in three possible verification strategies. In what follows, we refer to the object (resp. subject) to which a tag is assigned as the *target* object (resp. subject) of the tag. The first possible verification strategy is *issuer verification* of tags, where the system to which the issuer of a tag belongs is trusted for checking the validity of the tag. The second strategy is *target verification*, which enables systems to verify the validity of the tags assigned to an object according to the intention of the object's owner. A possible application scenario for target verification is the protection of digital media, where only the content owner is entitled to define the authorizations to access the object. Finally, *local verification* of tags can be employed in scenarios where there is no mutual trust among systems in different security domains. With local verification, the system which is interested in verifying the validity of a tag performs the check with respect to its local administrative policy.

The following example illustrates local verification of tags based on the administrative policy in Example 2.

Example 3 (Tag Verification). The French Navy distributes a document d containing the location of a suspected pirate attack to the other navies involved in Operation Atalanta. When d is received by the British Navy, it contains the following tags:

- $\mathsf{tag}(d)$ = {(fr_navy,secret),(s_4,inaccurate_information)}

where subject s_4 is labeled as follows:

- $\mathsf{tag}(s_4)$ = {(it_navy,reconnaissance_pilot)}

Since in Example 2 there is no rule defined by the British Navy that implies $\mathsf{allow}(s_4, d, assign\_tag, \{(s_4, inaccurate\_information)\})$, the tag added by subject s_4 is considered invalid. In fact, the policy of the British Navy allows only senior officers to tag documents marked as secret by an EU navy. The British Navy might thus decide to proceed with further investigations before concluding the inaccuracy of d's information.

For the sake of simplicity, the auditing mechanism discussed in this section verifies the compliance of tags with respect to the administrative policies that are currently in force. In other words, a tag is considered valid if its assignment is authorized by the administrative policy in force at the moment in which the tag is *verified*. An alternative verification mechanism could verify tags with respect to the administrative policy in force at the moment in which the tag was *assigned*. Intuitively, the implementation of

the latter mechanism is more complex and requires, e.g., timestamped tags and repositories containing all the administrative policies adopted by a system over time. An extension of the TBA^2 model in this direction is discussed in Section 6.

3.4 Tag Revocation

Whenever security administrators identify invalid tags, they should revoke them to preserve the consistency of function tag with respect to administrative policies. In a centralized system, revocation can be performed by simply deleting incorrect tags from the system. In a distributed system, revoking a set of tags is more complicated because it might not be possible to trace the tags issued by a given subject, and security administrators cannot delete tags assigned to subjects and objects residing in different security domains. TBA^2 enables a simple solution to this problem, where security administrators communicate the issuer-tag pairs to be revoked to other systems by broadcasting or publishing in an appropriate location *revocation lists* of tags. The recipient systems can then decide whether to revoke the listed tags or ignore the recommendation.

Definition 3 (Revocation List). *A revocation list is a triple $\langle s, so, T' \rangle$ where $T' \subseteq T$ is a set of tags, s is the issuer of the tags in T', and so is the target subject or object.*

Intuitively, a revocation list contains the set T' of tags assigned by a subject s to subject or object so which should be revoked according to the system publishing the revocation list.

Example 4 (Revocation List). The revocation list for the invalid tag identified by the British Navy in Example 3 is the following: $\langle s_4, d, \{inaccurate\_information\} \rangle$. The revocation list is published by the British Navy on its public record of invalid tags.

The decision of the security administrator of a system sys on whether to actually revoke the set of tags listed in a revocation list $\langle s, so, T' \rangle$ published by a system sys' is strictly correlated to the verification strategy employed by sys. If sys resorts to issuer (resp. target) verification, for instance, sys trusts the system to which s (resp. so) belongs to perform the validity check of tags T'. Consequently, if s (resp. so) belongs to sys', the security administrator of sys is likely to delete tags T' from its system. On the contrary, if sys resorts to local verification, it might decide to proceed with further investigations before deleting the tags. The revocation list published by the British Navy in Example 4, for instance, might be taken into consideration by the vessels of the British Navy, but ignored by the vessels of other EU countries, because derived through local verification with respect to the British Navy's policy.

As an alternative to revocation lists, we consider the use of *negative tags*. Negative tags are signed tags that state that a certain tag assigned to a subject or object is not (or no longer) valid according to the issuer of the negative tag. The advantage of negative tags is that they enable the verification of revoked tags, which might not be possible if the tags are deleted. However, this might lead to a very large number of tags (both "positive" and negative) assigned to each subject and object.

Rather than simply deleting invalid tags (or issuing negative tags), other approaches can be employed for restoring the compliance of tags with administrative policies. In

some critical or uncertain situation, for instance, security administrators might decide to simply highlight the invalid tags and refer to a competent user for determining what to do with them. Alternatively, systems may rely on *repair constraints* [11] to determine how to handle invalid tags. A repair constraint might, for example, dictate in which conditions cascading rather than non-cascade revocation should be applied on a tag.

4 Evaluation of TBA^2

In this section we evaluate the expressive power of the TBA^2 model. First, it is easy to demonstrate that TBA^2 is strictly more expressive than TBA. In fact, by not bounding the tags' issuers, TBA^2 can express exactly the same constraints definable by TBA. On top of this, associating an issuer to each tag enables the specification of authorization and administrative rules discriminating based on the issuer of tags, such as for instance linked roles in RT [18]. A linked role is a rule of the form $A.r \leftarrow B.r_1.r_2$, which states that subject A assigns a subject S_x (implicitly defined) to role r if S_x is labeled as a member of role r_2 by a subject S_y who is assigned to role r_1 by subject B. The reason why linked roles cannot be represented in TBA is that they require the binding of the subject of the first role r_1 to the issuer of the second role r_2. TBA^2 rules 1, 2, and 3 in Example 2 are examples of RT's linked roles.

We now show how TBA^2 can represent policies from two reference administrative models proposed in the literature, namely the Harrison, Ruzzo, and Ullman (HRU) model [13] and ARBAC97 [23]. The HRU model [13] employs an access matrix for the specification of the rights of users on the objects in a system, and relies on a set of commands for modifying users' authorizations. The model includes three predefined commands: commands CONFER and REVOKE allow the owner of an object to respectively grant to and revoke from other subjects any right on the objects she owns; command TRANSFER allows users to delegate their rights to other users. In TBA^2 we use $\mathsf{allow}(s, o, assign\_tag, T')$ to define commands CONFER and TRANSFER, and $\mathsf{allow}(s, o, revoke\_tag, T')$ to define the REVOKE command. We assume the tags in T to consist of pairs $\langle s, r\rangle$, representing each possible right $r \in R$ of a subject $s \in S$. The tags associated to an object define the rights of the users of a system on that object.

Example 5 (Mapping HRU to TBA^2). The following three rules define commands *CONFER*, *TRANSFER*, and *REVOKE* respectively:

1. $\mathsf{allow}(\mathrm{S}_x,\mathrm{O}_x,assign\_tag,\{(\mathrm{S}_y,\mathrm{R}_x)\})$:- $(sys,\langle \mathrm{S}_x,\mathrm{own}\rangle) \in \mathsf{tag}(O_x)$
2. $\mathsf{allow}(\mathrm{S}_x,\mathrm{O}_x,assign\_tag,\{(\mathrm{S}_y,\mathrm{R}_x)\})$:- $(sys,\langle \mathrm{S}_z,\mathrm{own}\rangle) \in \mathsf{tag}(O_x)$, $(\mathrm{S}_z,\langle \mathrm{S}_x,\mathrm{R}_x^*\rangle) \in \mathsf{tag}(O_x)$
3. $\mathsf{allow}(\mathrm{S}_x,\mathrm{O}_x,revoke\_tag,\{(\mathrm{S}_y,\mathrm{R}_x)\})$:- $(sys,\langle \mathrm{S}_x,\mathrm{own}\rangle) \in \mathsf{tag}(O_x)$

The first rule states that the owner S_x of an object O_x can assign any right R_x on O_x to any subject S_y. The tag representing the ownership of an object is a "system tag"; we use sys to denote the issuer of system tags. Rule 2 represents the right of a subject S_x to delegate a right R_x on object O_x to a subject S_y, provided that R_x is a transferable right (denoted by symbol * in the HRU model) assigned to S_x by the owner S_z of object O_x. Finally, rule 3 states that the owner of an object can revoke any right on that object.

Note that the HRU model allows users to define additional commands to modify the authorizations within a system. Since those commands are arbitrary, and are not described in the model, we cannot evaluate the expressiveness of TBA^2 with respect to them.

Next, we show how to represent in TBA^2 the administrative policies supported by ARBAC97 [23], an administrative model for role-based access control. In ARBAC97, roles are divided into two classes: *administrative roles* and *regular roles*. Both classes of roles are organized into hierarchies, where each role inherits all the rights assigned to the children nodes in the hierarchy. The ARBAC97 model relies on four commands for the specification of administrative policies:

1. $can\_assign(ar, \phi, \{rr_1, \ldots, rr_n\})$
2. $can\_revoke(ar, \{rr_1, \ldots, rr_n\})$
3. $can\_assignp(ar, \phi, \{rr_1, \ldots, rr_n\})$
4. $can\_revokep(ar, \{rr_1, \ldots, rr_n\})$

where ϕ (called prerequisite condition) is a boolean expression on regular roles, which defines the requirements on the membership (or non-membership) of a user to some roles. Commands (1) and (2) are used to specify the right to assign and revoke roles, while commands (3) and (4) define the rights to assign and revoke permissions. More precisely, command (1) defines the right of a member of the administrative role ar (or a member of an administrative role above ar in the hierarchy) to assign to a user who satisfies the prerequisite conditions ϕ the membership to regular roles $rr_1, \ldots, rr_n$. Command (2) assigns to members of the administrative role ar (or higher roles in the hierarchy) the right to revoke regular roles $rr_1, \ldots, rr_n$. Similarly, command (3) allows members of the administrative role ar (or higher) to assign to roles $rr_1, \ldots, rr_n$ any permission whose assignment to regular roles satisfies ϕ, and command (4) enables members of ar (or higher) to revoke any right to roles $rr_1, \ldots, rr_n$. To represent ARBAC97, we consider a set of administrative roles AR and regular roles RR to be defined as tags in T. The assignment of a user to a role is represented by the assignment of a tag from RR to the user. In addition, similarly to the previous example, we employ tags consisting of pairs $\langle s, r \rangle$ to represent a right $r \in R$ of a subject $s \in S$. Finally, for representing a prerequisite condition ϕ, we rewrite ϕ in disjunctive normal form, i.e., into a formula of the form $(CR_{11} \wedge \ldots \wedge CR_{1m_1}) \vee \ldots \vee (CR_{p1} \wedge \ldots \wedge CR_{pm_p})$, where CR_{ij} (with $i \in \{1, \ldots, p\}$, $j \in \{1, \ldots, m_i\}$) is either cr_{ij} or $\neg cr_{ij}$, with $cr_{ij} \in RR$, and the negation symbol $\neg$ denotes non-membership to a regular role. Negation as failure is employed to interpret negated roles: a user is not a member of a role cr_{ij} if she is not assigned a tag (s, cr_{ij}), for any $s \in S$.

Example 6 (Mapping ARBAC97 to TBA^2). The following TBA^2 rules define ARBAC97 commands (1), (2), (3), and (4) respectively:

1. $\mathsf{allow}(\mathrm{S}_x, \mathrm{S}_y, assign\_tag, \{(\mathrm{S}_x, \mathrm{rr}_1), \ldots, (\mathrm{S}_x, \mathrm{rr}_n)\})$:- $(*, \mathrm{ar}) \in \mathsf{tag}(\mathrm{S}_x)$, $(*, cr_{11}) \odot \mathsf{tag}(\mathrm{S}_y)$, $\ldots$, $(*, cr_{1m_1}) \odot \mathsf{tag}(\mathrm{S}_y)$

   $\ldots$

   $\mathsf{allow}(\mathrm{S}_x, \mathrm{S}_y, assign\_tag, \{(\mathrm{S}_x, \mathrm{rr}_1), \ldots, (\mathrm{S}_x, \mathrm{rr}_n)\})$:- $(*, \mathrm{ar}) \in \mathsf{tag}(\mathrm{S}_x)$, $(*, cr_{p1}) \odot \mathsf{tag}(\mathrm{S}_y)$, $\ldots$, $(*, cr_{pm_p}) \odot \mathsf{tag}(\mathrm{S}_y)$
2. $\mathsf{allow}(\mathrm{S}_x, \mathrm{S}_y, revoke\_tag, \{(*, \mathrm{rr}_1), \ldots, (*, rr_n)\})$:- $(*, \mathrm{ar}) \in \mathsf{tag}(\mathrm{S}_x)$

3. $\mathsf{allow}(\mathrm{S}_x,\mathrm{S}_y,assign\_tag,\{(\mathrm{S}_x,\langle \mathrm{S}_y,\mathrm{R}_x\rangle)\}) \text{ :- } (*,\mathrm{ar}) \in \mathsf{tag}(\mathrm{S}_x),\ (*,rr_1) \in \mathsf{tag}(\mathrm{S}_y),$
$(*,cr_{11}) \in \mathsf{tag}(S_{cr_{11}}),\ (*,\langle \mathrm{S}_{cr_{11}},\mathrm{R}_x\rangle) \odot \mathsf{tag}(O_{cr_{11}}),$
$\ldots,$
$(*,cr_{1m_1}) \in \mathsf{tag}(S_{cr_{1m_1}}),\ (*,\langle \mathrm{S}_{cr_{1m_1}},\mathrm{R}_x\rangle) \odot \mathsf{tag}(O_{cr_{1m_1}})$
$\ldots$
$\mathsf{allow}(\mathrm{S}_x,\mathrm{S}_y,assign\_tag,\{(\mathrm{S}_x,\langle \mathrm{S}_y,\mathrm{R}_x\rangle)\}) \text{ :- } (*,\mathrm{ar}) \in \mathsf{tag}(\mathrm{S}_x),\ (*,rr_n) \in \mathsf{tag}(\mathrm{S}_y),$
$(*,cr_{p1}) \in \mathsf{tag}(S_{cr_{p1}}),\ (*,\langle \mathrm{S}_{cr_{p1}},\mathrm{R}_x\rangle) \odot \mathsf{tag}(O_{cr_{p1}}),$
$\ldots,$
$(*,cr_{pm_p}) \in \mathsf{tag}(S_{cr_{pm_p}}),\ (*,\langle \mathrm{S}_{cr_{pm_p}},\mathrm{R}_x\rangle) \odot \mathsf{tag}(O_{cr_{pm_p}})$
4. $\mathsf{allow}(\mathrm{S}_x,\mathrm{S}_y,revoke\_tag,\{(*,\langle \mathrm{S}_y,\mathrm{R}_x\rangle)\}) \text{ :- } (*,\mathrm{ar}) \in \mathsf{tag}(\mathrm{S}_x),\ (*,rr_1) \in \mathsf{tag}(\mathrm{S}_y)$
$\ldots$
$\mathsf{allow}(\mathrm{S}_x,\mathrm{S}_y,revoke\_tag,\{(*,\langle \mathrm{S}_y,\mathrm{R}_x\rangle)\}) \text{ :- } (*,\mathrm{ar}) \in \mathsf{tag}(\mathrm{S}_x),\ (*,rr_n) \in \mathsf{tag}(\mathrm{S}_y)$

where $*$ indicates any subject in S, and $\odot$ is either $\in$ or $\notin$ depending on the corresponding element in ϕ. The first set of rules allows a subject S_x with administrative role ar to assign to a subject S_y whose roles satisfy the formula $(CR_{i1} \wedge \ldots \wedge CR_{im_i})$ (for any $i \in \{1,\ldots,p\}$) to regular roles $rr_1,\ldots,rr_n$. The second rule allows a member S_x of administrative role ar to revoke to a subject S_y regular roles $rr_1,\ldots,rr_n$, independently from the subject who assigned them. The set of rules in item 3 states that a subject S_x who is a member of administrative role ar may assign a right R_x to a subject S_y, provided that S_y is a member of regular role rr_j (with $j \in \{1,\ldots,n\}$), and R_x is a right whose assignment to regular roles satisfies $(CR_{i1} \wedge \ldots \wedge CR_{im_i})$. Finally, the set of rules in item 4 gives to a subject S_x having administrative role ar the right to revoke any right to the members of role rr_j.

In the example above we do not consider inheritance of rights among roles in a hierarchy. Rather, we assume that a rule is defined for each administrative role ar having a certain right. A role hierarchy could be easily defined using a predicate $higher\_role(ar_1, ar_2)$, and adding to each rule a condition $higher\_role(ar, ar_{min})$, where ar_{min} is the minimum role in the hierarchy to which the rule applies. In addition, we slightly modify the semantics of commands (3) and (4). Whereas in ARBAC97 permissions are assigned to roles, in our representation they are assigned to the members of a role. From the practical point of view, however, the two semantics are equivalent.

The examples above demonstrate that TBA^2 can express the administrative constraints defined by HRU and ARBAC97. As a matter of fact, neither HRU nor ARBAC97 fully exploit the expressiveness of TBA^2. As shown by Example 6, for instance, ARBAC97 does not exploit the capability of TBA^2 of constraining the issuer of a tag and the rights that a member of an administrative role may assign. With respect to the HRU model, TBA^2 allows for the specification of much more complex constraints than those defined in commands *CONFER*, *TRANSFER*, and *REVOKE*, e.g., based on the properties of subjects and objects. This implies that, in terms of expressive power, TBA^2 represents a more comprehensive solution than the considered models.

5 Related Work

TBA has been studied informally in [20, 26], though that work allows tags on subjects but not on objects. Next to it, substantial work has been done on logical access control

models, both based on Datalog (e.g., [2, 18, 22]) as well as on more expressive logics (e.g., [1, 9, 27]). While many of the existing logical access control languages can be used to encode tag-based authorization policies, it is the commitment to document and user tagging (an activity that can be carried out by users with a wide range of technical expertise) that makes TBA useful to a broad class of organizations.

The work related to the contributions of this paper spans two main topics: policy administration and auditing mechanisms. While many access control models for distributed systems have been proposed in the literature, policy administration received much less consideration. A number of administration models exist [4, 5, 10, 13, 17, 23], but they focus mainly on the expressivity of administrative policies, and do not consider the challenges associated with their enforcement in a distributed setting. The innovation of TBA^2 in this respect lies in the fact that it allows for an easy verification of policy compliance, thus not requiring entities in one security domain to trust systems in different security domains for the enforcement of administrative policy. In addition, we have shown that TBA^2 is more expressive than two reference administrative models, namely ARBAC97 [23] and HRU [13].

Similarly to TBA^2, the existing a posteriori solutions (e.g., [7]) perform the verification of policy compliance through auditing mechanisms. However, to achieve this, they rely on logging mechanisms that record users' actions, and trusted auditing authorities that verify the compliance of those actions with policies. Our model represents a lightweight solution for policy compliance verification that does not require the realization of such an auditing infrastructure. We propose the use of trust management algorithms [19, 25] to support the verification of policy compliance.

6 Discussion and Conclusions

In this paper we have introduced TBA^2, an extension of the TBA model [15] that enables policy administration in distributed systems. Similarly to TBA, TBA^2 allows relatively untrained users to assign descriptive tags to a system's subjects and objects; trained security experts then write logic-based authorization policies that define access rights in terms of those tags. In addition, by linking each tag to its issuer (i.e., the user who assigned it), TBA^2 enables the specification of fine-grained administrative policies whose enforcement can be verified through a lightweight auditing technique. We have shown that our model is more expressive than TBA and than the HRU [13] and ARBAC97 [23] administrative models. Thus, TBA^2 represents a flexible, easy to use, yet expressive access control solution which matches the needs of real-world organizations.

The auditing mechanism proposed in Section 3.3 verifies tags' validity with respect to the administrative policy currently in force within a system. In some situations, however, it is preferable to verify the validity of a tag with respect to the administrative policies effective when the tag was issued. For example, assume that the commanding officer of a British Navy vessel is summoned by the EU for a meeting at the Operation Atalanta's headquarter. Then, the commanding officer would have to temporarily delegate the command of the vessel and the deriving responsibilities and authorizations to another officer until her return. During this period, the appointed officer will have to take several decisions which might lead to the granting and revocation of authorizations

to the vessel's operators and to the tagging of several data objects exchanged among the collaborating navies. With the verification mechanism presented in Section 3.3, the revocation of the officer's rights by the commanding officer upon her return would have the undesirable effect of invalidating all the authorizations and tags assigned by the officer during her command. The design of an auditing mechanism verifying tags' validity with respect to the administrative policy in force when a tag was assigned would require two main extensions to the TBA^2 model. First, it would require the association of a timestamp to each tag to demonstrate when it was issued. Second, all the administrative policies employed by a system during its lifetime would need to be stored in a repository, together with the time interval in which they were effective. Then, whenever a tag needs to be verified, its timestamp can be used to retrieve from the repository the policy that was in force when the tag was issued, against which the validity check must be performed. The resulting enforcement mechanism is similar to those used for the enforcement of history-based access control policies [16].

To conclude, we point out that the model proposed in this paper enables security administrators to verify the compliance of users' actions with respect to the administrative policies in force within a system, but provides no guarantee that these policies are correctly specified. The verification of administrative policies with respect to the desired security properties of a system can be achieved through model checking techniques [28]. Finally, we argue that even though TBA^2 is presented as an access control solution for distributed systems, also centralized systems would benefit from employing the model. In fact, the association of each tag to its issuer enhances the "observability" of user's actions, simplifying the detection of policy violations, and may be used as a discriminant by other users in the system to determine whether a certain tag should be considered valid. Signed tags are currently employed by several existing web applications and social networks (e.g., Facebook).

Acknowledgments This work has been done in the context of the THeCS project, which is supported by the Dutch national program COMMIT. Adam J. Lee was supported in part by the US National Science Foundation under awards CNS-0964295 and CNS-1228697.

References

1. M. Abadi, M. Burrows, and B. Lampson. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.
2. M. Y. Becker, C. Y. Fournet, and A. D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.
3. D. E. Bell. Looking Back at the Bell-La Padula Model. In *Proceedings of ACSAC'05*, pages 337–351. IEEE Computer Society, 2005.
4. M. Ben-Ghorbel-Talbi, F. Cuppens, N. Cuppens-Boulahia, and A. Bouhoula. Revocation Schemes for Delegation Licences. In *Proceedings of ICICS'08*, LNCS 5308, pages 190–205. Springer, 2008.
5. M. Ben-Ghorbel-Talbi, F. Cuppens, N. Cuppens-Boulahia, and A. Bouhoula. A delegation model for extended RBAC. *Int. J. Inf. Sec.*, 9(3):209–236, 2010.
6. E. Bertino, P. Samarati, and S. Jajodia. An Extended Authorization Model for Relational Databases. *IEEE Trans. Knowl. Data Eng.*, 9(1):85–101, 1997.

7. J. G. Cederquist, R. Corin, M. A. C. Dekker, S. Etalle, J. I. den Hartog, and G. Lenzini. Audit-based compliance control. *Int. J. Inf. Sec.*, 6(2):133–151, 2007.
8. J. Crampton and H. Khambhammettu. Delegation in role-based access control. *Int. J. Inf. Sec.*, 7(2):123–136, 2008.
9. J. Crampton, G. Loizou, and G. Oshea. A logic of access control. *The Computer Journal*, 44(1):137–149, 2001.
10. M. Dekker, J. Crampton, and S. Etalle. RBAC administration in distributed systems. In *Proceedings of SACMAT'08*, pages 93–102. ACM, 2008.
11. G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Trans. Knowl. Data Eng.*, 15(6):1389–1408, 2003.
12. P. P. Griffiths and B. W. Wade. An authorization mechanism for a relational database system. *ACM Trans. Database Syst.*, 1(3):242–255, 1976.
13. M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.
14. S. Heeps, J. Sventek, N. Dulay, A. S. Filho, E. Lupu, M. Sloman, and S. Strowes. Dynamic Ontology Mapping for Interacting Autonomous Systems. In *Proceedings of IWSOS'07*, LNCS 4725, pages 255–263. Springer, 2007.
15. T. L. Hinrichs, W. C. Garrison III, A. J. Lee, S. Saunders, and J. C. Mitchell. TBA: A Hybrid of Logic and Extensional Access Control Systems. In *Proceedings of FAST'11*, LNCS. Springer, 2011.
16. H. Koshutanski, F. Martinelli, P. Mori, and A. Vaccarelli. Fine-grained and History-based Access Control with Trust Management for Autonomic Grid Services. In *Proceedings of ICAS'06*, pages 34–43. IEEE Computer Society, 2006.
17. N. Li and Z. Mao. Administration in role-based access control. In *Proceedings of ASIACCS'07*, pages 127–138. ACM, 2007.
18. N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a Role-Based Trust-Management Framework. In *Proceedings of S&P'02*, pages 114–130. IEEE Computer Society, 2002.
19. N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, 2003.
20. M. Najafian Razavi and L. Iverson. Supporting selective information sharing with people-tagging. In *Proceedings of CHI'08*, pages 3423–3428. ACM, 2008.
21. S. Osborn, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Trans. Inf. Syst. Secur.*, 3(2):85–106, 2000.
22. C. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes. SPL: An access control language for security policies with complex constraints. In *Proceedings of NDSS'01*, 2001.
23. R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1):105–135, 1999.
24. D. Trivellato, F. Spiessens, N. Zannone, and S. Etalle. Reputation-Based Ontology Alignment for Autonomy and Interoperability in Distributed Access Control. In *Proceedings of CSE'09*, volume 3, pages 252–258. IEEE Computer Society, 2009.
25. D. Trivellato, N. Zannone, and S. Etalle. GEM: a Distributed Goal Evaluation Algorithm for Trust Management. *Journal of Theory and Practice of Logic Programming*, 2012. To appear.
26. Q. Wang, H. Jin, and N. Li. Usable Access Control in Collaborative Environments: Authorization Based on People-Tagging. In *Proceedings of ESORICS'09*, LNCS 5789, pages 268–284. Springer, 2009.
27. D. Wijesekera and S. Jajodia. Policy algebras for access control - the predicate case. In *Proceedings of CCS'01*, pages 171–180. ACM, 2001.
28. N. Zhang, M. Ryan, and D. P. Guelev. Synthesising verified access control systems through model checking. *Journal of Computer Security*, 16(1):1–61, 2008.