Dynamics in Delegation and Revocation Schemes: A Logical Approach

Guillaume Aucher[1], Steve Barker[2], Guido Boella[3], Valerio Genovese[3,4], and Leendert van der Torre[4]

[1] University of Rennes 1 - INRIA, France
[2] King's College London, UK
[3] University of Torino, Italy
[4] University of Luxembourg, Luxembourg

Abstract. In this paper we first introduce a logic for describing formally a family of delegation and revocation models that are based on the work in Hagström *et al.*. We then extend our logic to accommodate an epistemic interpretation of trust within the framework that we define. What emerges from this work is a rich framework of formally well-defined delegation and revocation schemes that accommodates an important trust component.

1 Introduction

Delegation and revocation are broad concepts that are fundamentally important in modelling and reasoning about (dynamic) distributed systems. In the context of multi-agent systems (MAS), delegation is important in relation to the coordination of agents and for the coordinating of activities within organizational structures [5]. Trust is, in turn, a fundamental notion in delegation and revocation; ordinarily, a principal i may delegate an access privilege a on an object o to a principal j iff i trusts j sufficiently not to abuse the trust i has in j to perform the action a in relation to o. In the context of revocation, it is when i loses trust in j, in relation to exercising the privilege a on o, that i revokes the a privilege on o from j. Although the importance of the trust dimension has been recognized in delegation-revocation, it is our contention that more work is required on the formal specification and reasoning about trust in the context of delegation and revocation. In this paper, our focus is on formally defining a general, dynamic delegation-revocation framework that accommodates an important aspect of trust. A feature of MAS is that agents are autonomous and therefore they can act with respect to a subjective perception of the environment. For instance, a verifier may decide not to concede access to agents that she does not trust or that have been delegated by other untrusted agents. In relation to this observation, in this paper we contribute to the study of delegation and revocation in the context of distributed systems, and multi-agent systems in particular, by addressing the following key research question: *How to define a formal framework to model and reason about delegation and revocation in the context of multi-agent (and*

other distributed) systems? This generally stated question breaks down into at least three important sub-questions that we intend to address: *How to update privileges on objects in a dynamic, multi-agent environment? How to specify and reason about different types of delegation and revocation schemes? How to study delegation of access privileges when trust interferes with the fact that an agent has been permitted to access?*

Our main question and each of the subquestions that we consider have been considered in the past, but the novelty of what we describe is to be understood in terms of the new formal approaches that we introduce to address them. The need for formal representations of security concepts is well understood (e.g., formal representations of security concepts are important for constructing assurance proofs). Our work is also motivated by the more specific observations that distributed access control systems can be seen as a type of a multi-agent system for which delegation models in "classical" security need to be extended. We need to also *use* our logical framework to reason about delegation-revocation policies and we require fast and effective tools for that. Delegation is an intrinsically dynamic process, therefore we additionally need to define dynamic operators that formalize a range of delegation-revocation schemes. The explicit representation of trust that we accommodate requires us to face two challenges: first, how to make the verifier autonomous to decide whether to give access in case of there being authorized but untrusted agents. Second, how to generalize the revocation policies of Hagström *et al.* [9] by considering whether an agent who delegated a permission is trusted or not. We address all of these issues in this paper.

The methodology that we employ in addressing these issues can be understood in the following way. First, we show that our framework can embody delegation and revocation schemes as addressed by the distributed access control community. In particular, we model all of the revocation schemes that are semi-formally introduced in [9] by using a dynamic variant of propositional logic. The work in [9] is among the most general models to handle dynamics in delegation chains and is the basis of several applied delegation models in security (Section 5 of [9]). Second, we extend the proposed framework to study relationships between trust and privilege delegation by explicitly modeling beliefs about trust relationships among agents.

Our contributions on these things can be summarized thus: (i) we formalize, in logic, the Hagström *et al.* framework (in [9], a semi-formal account is provided), (ii) we demonstrate the translation of our logic into "programs" (a notion that we will define later) that describe the effects of performing delegation and revocation actions, and (iii) we describe an extended form of our logic that allows for representing and reasoning about the beliefs that agents have of principals in a distributed delegation-revocation framework.

In Section 2, we describe a general authorization system, along the lines of [9], and we give some basic definitions. In Section 3, we introduce the logic that we use in order to represent formally the range of delegation and revocation schemes, of the Hagström *et al.* type, that we consider. In Section 4, we describe the use of our logic for representing delegation policies and, in Section 5, we

describe the use of our logic for representing revocation schemes. In Section 6, we make the key move of extending the formalization of policies expressible in the Hagström *et al.* framework to accommodate an epistemic logic of trust. The latter is used to account for reasoning about belief and trust in the delegation-revocation context. In Section 7, we describe related work and, in Section 8, we draw conclusions and make some suggestions for further work.

2 System Description

In this Section, we formalize the general concepts and notation introduced informally in [9]. The notation is intended to represent a generic access control framework using an ownership-based model with grant option for both positive and negative permissions, and where negative permissions dominate positive ones. We draw the reader's to a simplified version of the distributed authoriza-

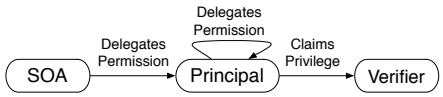

Fig. 1. The Authorization Model

Fig. 2. Dominance Relation $\mathcal{R}$

tion model described in [1, 8], and illustrated in Figure 1, where an agent receives a privilege, directly or indirectly, from a *source of authority* (SOA). The SOA is an agent that has full power over a resource and is the ultimate authority w.r.t. accesses to that specific resource. The verifier is a particular agent in charge of checking whether another agent, who received a privilege directly or indirectly from the SOA, that wants to exercise an access permission is authorized so to do.

2.1 Basics

Let $\mathbb{AG}$ be a finite set of *agents* (users) in the authorization system. Let $\mathbb{O}$ be the (finite) set of system *objects* for which authorizations can be stated. Finally, let $\mathbb{A}$ be the (finite) set of *accesses* over objects; by accesses we mean the actions that agents may perform on objects. We assume that all authorizations in the system are stored in an authorization specification $AUTH$, and that every authorization is of the form $(i, j, (a, o), alp, dlp)$ where, i and j are two agents, the *grantor* and the *subject*; (a, o) is an *access type* and specifies an action a on a object o; $alp \in \{\top, \bot\}$ (access level permission) is a flag which specifies if the authorization is an access level permission $alp = \top$ or an access level denial $alp = \bot$; $dlp \in \{\top, \bot\}$ (delegation level permission) is a flag which specifies whether or not the authorization gives to j the authority to further delegate the permission. For instance, the authorization $(i, j, (a, o), \top, \top) \in AUTH$ says that agent i gives agent j the permission to perform action a on object o and

the authority to further delegate this permission to other principals. On the contrary $(i, j, (a, o), \top, \bot) \in AUTH$ means that j still gets from i the permission to perform the a action on o but she is not granted by i to further delegate the permission. In case of an authorization being a denial, i.e., $alp = \bot$, we require also dlp to be $\bot$ in order to represent that the agent cannot delegate a permission to access what she does not have herself. Hence, in our model we have three possible permissions (i.e., $(\top, \top)$, $(\top, \bot)$, $(\bot, \bot)$).

Definition 1 (Positive and Negative Permissions). *Given an authorization $(i, j, (a, o), alp, dlp)$, we refer to (alp, dlp) as a* negative permission *if $alp = \bot$; otherwise, we call it a* positive permission.

When a user receives both a positive and a negative permission for the same $(action, object)$ pair, there is a "conflict" between the two assignments. Hence, the set of permissions is divided into one set of *active permissions* and one set of *inactive permissions*. Active permissions can be inactivated when a negative permission is granted (e.g., during a revocation). Inactive permissions, instead, can be activated when a negative permission for the same target is removed.

In Figure 2, we illustrate a dominance relation $\mathcal{R}$ between permissions such that if $(alp, dlp)\mathcal{R}(alp', dlp')$ reads as, if an agent i has permission (alp, dlp) then it can grant an authorization of type $(i, _, (_, _), alp', dlp')$. Intuitively, $(alp, dlp)\mathcal{R}(alp', dlp')$ means that permission (alp, dlp) is stronger than (alp', dlp').

In line with [9], we require an authorization specification to satisfy the following property:

Definition 2 (Connectivity Property). *For all authorizations in $AUTH$, if an agent i is the grantor of a permission (alp, dlp) for permissions target (a, o) to the subject j, then i must have a permission (alp', dlp') such that $(alp', dlp')\mathcal{R}(alp, dlp)$.*

The connectivity property can be considered as a constraint over the authorization specification $AUTH$. Intuitively, it assures that if an agent i delegates a permission (alp, dlp) to j for the access type (a, o) then she has the permission to do so.

Definition 3 (Delegation Chain). *Given an access type (a, o), a* delegation chain *$[x_1, x_2, \ldots, x_n]_{(a,o)}$ is a sequence of authorizations of the form $(x_1, x_2, (a, o), alp_1, dlp_1)$, $\ldots, (x_{n-1}, x_n, (a, o), alp_n, dlp_n)$.*

An agent j is granted the access type (a, o) if and only if the verifier can check the existence of a *rooted delegation chain*, which we define next.

Definition 4 (Rooted Delegation Chain). *A delegation chain $[x_1, x_2, \ldots, x_n]_{(a,o)}$ is* rooted *if and only if the following hold: x_1 is a source of authority for object o; all agents $x_2, \ldots, x_{n-1}$ have an active privilege $(\top, \top)$ for access type (a, o); agent x_n has an active privilege $(\top, dlp)$ with $dlp \in \{\top, \bot\}$.*

The notion of rooted delegation chain is pivotal because it corresponds to the notion of permission in standard access control. In this view, the connectivity

property assures that if $(i, j, (a, o), alp, dlp) \in AUTH$ then there is a rooted delegation chain that links j to a source of authority for o. In [9], Hagström *et al.* impose the above property to hold in *any* authorization specification. However, in highly distributed scenarios (e.g., GRID systems) it may be extremely difficult to enforce the connectivity property *a priori* for every access type (a, o) (see [8] for an example). In Section 3, we relax this requirement and we give a formal account of the properties reported above in order to check whether a node in the authorization specification is part of a rooted chain.

As reported in [9], the chains of granted authorizations in a system can be represented by directed graphs. The nodes contain information about subject, object and access type, and the arcs are labelled with the granted permission (alp, dlp). There is an arc from node $(i, (a, o))$ to node $(j, (a, o))$ if there is an entry in the authorization specification with $(i, j, (a, o), alp, dlp)$. An arc from node i to node j is labelled with the permission granted by user i to user j.

Active arcs have unbroken lines and inactive arcs have dashed lines to indicate that although they are still in $AUTH$, they are not in effect because they have been overruled by a negative permission.

3 The Logic

We extend the propositional language *Prop* with dynamic operators to specify programs that update an authorization specification by issuing (or revoking) credentials certificates.

Definition 5 (Syntax). *We define inductively the language $\mathcal{L}$ as follows:*

$$\varphi ::= p \mid \varphi \wedge \varphi \mid \neg\varphi \mid [\pi]\varphi \qquad \pi ::= +p \mid -p \mid \varphi? \mid \pi \cup \pi \mid \pi;\pi$$

where p ranges over $\Phi = \{soa_{i,o}, (i,(a,o),j)^D_+, (i,(a,o),j)^D_-, (i,(a,o),j)^P_+, (i,(a,o),j)^P_- \mid a \in \mathbb{A}, o \in \mathbb{O}, i, j \in \mathbb{AG}\}$, with $\mathbb{A}, \mathbb{O}$ and $\mathbb{AG}$ being finite sets.

The propositional atoms in Φ describe the state of the authorization system. $soa_{i,o}$ reads as: "agent i is the source of authority over object o". To describe the steps of delegation chains we use triples such that $(i,(a,o),j)^P_+$ (resp. $(i,(a,o),j)^D_+$) reads as: "there is a certificate supporting that i delegates an access (resp. delegation) level permission to j" while $(i,(a,o),j)^P_-$ (resp. $(i,(a,o),j)^D_-$) reads as: "there is a certificate supporting that i gives a negative access (resp. delegation) level permission".

Given a generic dynamic formula $[\pi]\varphi$ we read it as follows: "after executing program π, the formula φ holds true". A program is therefore intended as a sequence of instructions such that : $[+p]\varphi$ (resp. $[-p]\varphi$) reads as: "after making p true (resp. false), φ holds"; $[\varphi?]\psi$ reads as: "If φ is true, then ψ is the case"; $[\pi \cup \pi']\varphi$ reads as: "after executing π, φ holds *and*, after executing π', φ holds"; $[\pi;\pi']\varphi$ reads as: "After executing π and then π', φ holds". For readability, we adopt the following abbreviations: $(i,(a,o),j)^{P,D}_+ \stackrel{\text{def}}{=} (i,(a,o),j)^P_+ \wedge (i,(a,o),j)^D_+$; $\neg(i,(a,o),j)^{P,D}_- \stackrel{\text{def}}{=} \neg(i,(a,o),j)^P_- \wedge \neg(i,(a,o),j)^D_-$; `if` φ `then` τ_1

`else` $\tau_2 \stackrel{\text{def}}{=} ((\varphi?;\tau_1) \cup (\neg\varphi?;\tau_2))$; `for all` $(x \in \{s_1,\ldots,s_n\})$ `do` $\tau_1(x)$ `end for` $\stackrel{\text{def}}{=} \tau_1(s_1);\ldots;\tau_1(s_n)$.

Definition 6 (Semantics). *A valuation Θ is a function assigning a truth value to each propositional atom: $\Theta : \Phi \to \{\top,\bot\}$. Given a valuation Θ of propositional logic and $p \in \Phi$, the updates Θ^{+p} and Θ^{-p} are defined as follows.*

$$\Theta^{+p}(q) = \begin{cases} \top & \text{if } p = q, \\ \Theta(q) & \text{otherwise.} \end{cases} \quad \Theta^{-p}(q) = \begin{cases} \bot & \text{if } p = q, \\ \Theta(q) & \text{otherwise.} \end{cases}$$

Let Θ be a valuation and $\phi \in \mathcal{L}$. The satisfaction relation $\Theta \models \phi$ is defined inductively as follows (we omit $\neg$ and $\wedge$).

$$\begin{array}{llll} \Theta \models p & \text{iff } \Theta(p) = \top & \Theta \models [\psi?]\phi & \text{iff } \Theta \models \psi \to \phi \\ \Theta \models [+p]\phi & \text{iff } \Theta^{+p} \models \phi & \Theta \models [\pi;\pi']\phi & \text{iff } \Theta \models [\pi][\pi']\phi \\ \Theta \models [-p]\phi & \text{iff } \Theta^{-p} \models \phi & \Theta \models [\pi \cup \pi']\phi & \text{iff } \Theta \models [\pi]\phi \wedge [\pi']\phi \end{array}$$

We exploit our basic dynamic operators to model certificate creation (granting) and deletion (revoking) by defining the following programs: $i \xrightarrow{(a,o)}_D j \stackrel{\text{def}}{=} +(i,(a,o),j)^D_+; i \xleftarrow{(a,o)}_D j \stackrel{\text{def}}{=} -(i,(a,o),j)^D_+; i \xrightarrow{(a,o)}_P j \stackrel{\text{def}}{=} +(i,(a,o),j)^P_+; i \xleftarrow{(a,o)}_P j \stackrel{\text{def}}{=} -(i,(a,o),j)^P_+; i \xrightarrow{-(a,o)} j \stackrel{\text{def}}{=} +(i,(a,o),j)^P_-; +(i,(a,o),j)^D_-; i \xleftarrow{-(a,o)} j \stackrel{\text{def}}{=} -(i,(a,o),j)^P_-; -(i,(a,o),j)^D_-$. For instance, $i \xrightarrow{(a,o)}_D j$ reads as: "a certificate supporting that i grants j the authority to delegate (a,o) is issued." while $i \xrightarrow{-(a,o)}_P j$ reads as: "a certificate supporting a negative permission granted by i to j for (a,o) is issued".

Next, we define the logic that we use for our delegation-revocation framework.

Definition 7. *The logic L is defined by the following axiom schemes and inference rules.*

Taut	$\vdash \phi$ *for all propositional tautologies* ϕ *based on* Φ	
K+	$\vdash [+p](\phi \to \psi) \to ([+p]\phi \to [+p]\psi)$	
K-	$\vdash [-p](\phi \to \psi) \to ([-p]\phi \to [-p]\psi)$	
Det+	$\vdash \neg[+p]\phi \leftrightarrow [+p]\neg\phi$	
Det-	$\vdash \neg[-p]\phi \leftrightarrow [-p]\neg\phi$	
Test	$\vdash [\psi?]\phi \leftrightarrow (\psi \to \phi)$	
Red1	$\vdash [+p]p$	
Red2	$\vdash [+p]q \leftrightarrow q$	*if* $p \neq q$
Red3	$\vdash [-p]\neg p$	
Red4	$\vdash [-p]q \leftrightarrow q$	*if* $p \neq q$
Comp	$\vdash [\pi;\pi']\phi \leftrightarrow [\pi][\pi']\phi$	
Choice	$\vdash [\pi \cup \pi']\phi \leftrightarrow [\pi]\phi \wedge [\pi']\phi$	
Nec	*If* $\vdash \phi$ *then* $\vdash [+p]\phi$ *and* $\vdash [-p]\phi$	
MP	*If* $\vdash \phi$ *and* $\vdash \phi \to \psi$ *then* $\vdash \psi$	

Proposition 1. *For all formula $\phi \in \mathcal{L}_D$, there is $Red(\phi) \in Prop$ such that $\vdash \phi \leftrightarrow Red(\phi)$. The reduction of ϕ to $Red(\phi)$ is polynomial in the size of ϕ*

Proof (sketch). We prove it by successive inductions. We use in great extent the 'reduction' axioms K+, K-, Det+, Det-, Test, Red1 to Red4: they all 'push through' the connectives, except for the basic cases Test and Red1-Red4 where the dynamic modalities [+p] and [-p] disappear.

The above proposition is extremely important because it shows that *every* dynamic formula of the type $[\pi]\varphi$ can be reduced in an *equivalent* static formula in standard propositional logic. As a consequence of Proposition 1 we get the following theorem:

Theorem 1. *The semantics of $\mathcal{L}_D$ is sound and complete w.r.t. the logic L. The logic L is also decidable and NP-complete.*

Definition 8 (Rooted Delegation Chain). *In the system represented by a valuation Θ, there is a rooted delegation chain ending at the node $(j,(a,o))$ iff $\Theta \models \mathcal{CP}_{\emptyset}(j,(a,o))$, where*

$$\begin{aligned}&\mathcal{CP}_S(j,(a,o)) = \\ &\quad \bigvee_{i\notin S}\big(((i,(a,o),j)^P_+ \wedge \neg(i,(a,o),j)^P_- \wedge soa_{i,o})\vee \\ &\quad ((i,(a,o),j)^P_+ \wedge \neg(i,(a,o),j)^P_- \wedge \mathcal{CP}^{P,D}_{S\cup\{i,j\}}(i,(a,o))\big)\end{aligned}$$

$$\begin{aligned}&\mathcal{CP}^{P,D}_S(j,(a,o)) = \\ &\quad \bigvee_{i\notin S}\big(((i,(a,o),j)^{P,D}_+ \wedge \neg(i,(a,o),j)^{P,D}_- \wedge soa_{i,o})\vee \\ &\quad ((i,(a,o),j)^{P,D}_+ \wedge \neg(i,(a,o),j)^{P,D}_- \wedge \mathcal{CP}^{P,D}_{S\cup\{i,j\}}(i,(a,o))\big)\end{aligned}$$

Intuitively, $\mathcal{CP}_S(j,(a,o))$ reads as: "There is a rooted delegation chain (with no agent in S) such that j is granted an access level permission (i.e., $alp = \top$) for (a,o)". Notice that our definition of $\mathcal{CP}_S(j,(a,o))$ is well-founded because we have a finite number of agents, object and actions.

An authorization that has the connectivity property as reported in Definition 2 can be seen as a particular valuation which complies with the following definition.

Definition 9 (Connectivity Property). *A system represented by a valuation Θ has the connectivity property iff for all access types (a,o), $\Theta \models \mathcal{CP}(a,o)$, where* $\mathcal{CP}(a,o) = \bigwedge_{i\in\mathbb{AG}}\big((i,(a,o),j)^P_+ \rightarrow \mathcal{CP}_{\emptyset}(i,(a,o))\big)$

We now introduce two notions that are pivotal in formally defining the revocation schemes presented in Section 5.

Definition 10 (Independency). *In a system represented by Θ, given a subject j with a permission (alp, dlp) for access type (a,o), j is said to be* independent *of a subject i iff $\Theta \models \mathcal{CP}_{\{i\}}(j,(a,o))$*

Definition 11 (Reachability). *In a system represented by a valuation Θ we say that j is reachable from i via a delegation chain for access type (a,o) iff $\Theta \models R_{\emptyset}(j,i,(a,o))$ where* $R_S(j,i,(a,o)) = (i,(a,o),j)^P_+ \vee \bigvee_{x\notin S}\big((x,(a,o),j)^{P,D}_+ \wedge R_{S\cup\{x\}}(x,i,(a,o))\big)$[1]

[1] Notice that we do not check for the arc in the delegation chain to be active.

Automated Theorem Proving As shown in Proposition 1, the logic defined above is sound and complete w.r.t. propositional logic. In order to show how to use state of the art theorem provers to reason about delegation and revocation schemes, we developed a parser (written in SCHEME) which implements a set of complete reduction axioms and translates dynamic formulas, as reported in Definition 5, into (static) propositional logic. The parser translates a set of formulas written in our logical framework into first-order formulas compatible with SPASS [15] syntax. Due that our language is finite, SPASS automatically instantiates the translated formulas into propositional logic and then uses a SAT solver to check satisfiability[2].

4 Delegation Schemes

As pointed out in [8], in the information security literature, *delegation* normally describes the act of distributing privileges to agents in distributed systems. In general, there are two possible kinds of delegation:

Delegation as creation of new privilege: the delegatee receives its own privilege which is independent of the delegator's privilege in the sense that if the delegator's privilege is revoked, then it does not necessarily mean that the delegatee's privilege is revoked. A special case is the *transfer* of a new privilege, which models the creation of a new privilege and a revocation of an old one;

Delegation by proxy: The delegatee does not receive its own privilege, but can exercise the privilege through the delegator, in the sense that the delegator *speaks for* or *acts on behalf of* the delegator.

On the first type of delegation, an agent i has a direct privilege to act on an object o if she is the SOA for it (i.e., $soa_{i,o}$). To model delegation by proxy instead, we need to keep track of the delegation chains (represented through atoms like $(i,(a,o),j)_{+}^{P,D}$) on which an agent depends for a given privilege.

We can accommodate the different types of delegation by exploiting the dynamic operators defined in the previous section. For instance, we can model delegation as creation of new privileges with the following programs: "Agent i assigns (if she has the power) a new privilege on object o to agent j": (`if` $soa_{i,o}$ `then` $+soa_{j,o}$); "Agent i transfers her privilege over o to agent j": (`if` $soa_{i,o}$ `then` $-soa_{i,o}; +soa_{j,o}$).

5 Revocation Schemes

In this section, we define the revocation operations that are informally described in [9]. The following schemes are sufficiently general to model a great deal of real-world distributed authorization architectures. The main contribution of this section is that for each revocation scheme S we define a program π_S such that we read $[\pi_S]\varphi$ as: "after the execution of a revocation operation S, φ holds".

[2] The parser is available at `http://www.di.unito.it/~genovese/tools/delegation2spass.zip`

Due to space constraints, we refer to $\pi_S[n-m]$ as the block instructions from line n to m of the program π_S.

As in [9], we divide revocation schemes into *positive* and *negative*, depending on the revocation action of deleting a certificate or of issuing a negative permission.

When i revokes a permission to j, we identify two types of agents: (i) those that are not independent from i and delegated the same permission to j (see Definition 10) and (ii) those that are reachable from j (see Definition 11) in the delegation chain. We classify a revocation operation as *weak/strong* and *local/global*, depending on how it influences agents of type (i) and (ii). A revocation operation is *weak* (resp. *strong*) if, in revoking a permission from i to j, none (resp. all) of the agents of type (i) are forced to revoke their delegation. Instead, we classify a revocation operation as *local* (resp. *strong*) if the algorithm influences none (resp. all) of the agents of type (ii).

An important property of *all* the programs implementing the revocation schemes is as follows

Theorem 2 (Invariance under connectivity). *After the execution of* any *program implementing the revocation schemes, the resulting delegation chain satisfies the connectivity property.*

5.1 Positive Revocation Schemes

Weak Local Delete The *weak local delete* operation is the simplest form of revocation. After the application of the weak local delete operation on a permission (alp, dlp) for a given access type (a, o) granted by agent i to j, the following three post-conditions must be satisfied [9]: i no longer grants j the permission (alp, dlp); Permissions for (a, o) granted to j by users other than i are intact; Permissions for subjects other than j are intact. However, the grantors of permissions for users directly following j in the graph for (a, o) may have changed in order for the connectivity property to be satisfied;

In Figure 4 we show the resulting delegation chain after the execution of program $WLD_{i,j}$.

Strong Local Delete The application of the *strong local delete* operation on a permission (alp, dlp) for access type (a, o) granted by agent i to agent j has to satisfy the following post-conditions: i no longer grants j the permission (alp, dlp); Permissions for access type (a, o) granted to j by every agent z other than i are intact if they are independent of i. Otherwise, they are restricted to satisfy the connectivity property for those paths from z that are independent of i; Positive (and negative) permissions for agents other than j are intact. However, the grantors of permissions for agents directly following j in the graph for (a, o) may have changed in order for the connectivity property to be satisfied. In Figure 6, we show the resulting delegation chain after the execution of program $SLD_{i,j}$.

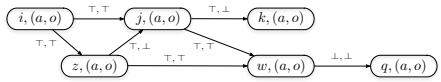

Fig. 3. A Delegation Chain

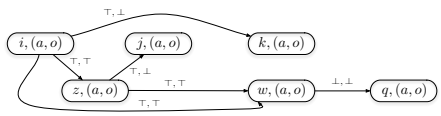

Fig. 4. Weak Local Delete

1: $i \xleftarrow{(a,o)}_{D,P} j$;
2: **for all** $k \in \mathbb{AG}$ **do**
3: **if** $((j,(a,o),k)^{P}_{+} \wedge \neg\mathcal{CP}_{\emptyset}(j,(a,o)))$ **then**
4: $j \xleftarrow{(a,o)}_{P} k$;
5: **if** $(\neg\mathcal{CP}_{\emptyset}(k,(a,o)))$ **then**
6: $i \xrightarrow{(a,o)}_{P} k$;
7: **end if**
8: **end if**
9: **if** $((j,(a,o),k)^{D}_{+} \wedge \neg\mathcal{CP}_{\emptyset}(j,(a,o)))$ **then**
10: $j \xleftarrow{(a,o)}_{D} k$;
11: **if** $(\neg\mathcal{CP}_{\emptyset}(k,(a,o)))$ **then**
12: $i \xrightarrow{(a,o)}_{D} k$;
13: **end if**
14: **end if**
15: **end for**

Fig. 5. $WLD_{i,j}$ Program

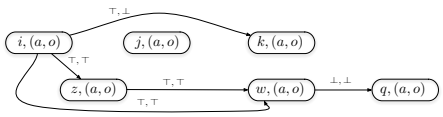

Fig. 6. Strong Local Delete

1: $i \xleftarrow{(a,o)}_{P,D} j$;
2: **for all** $x \in \mathbb{AG}$ **do**
3: **if** $((x,(a,o),j)^{P}_{+} \wedge \neg\mathcal{CP}_{\{i\}}(x,(a,o)))$ **then**
4: $x \xleftarrow{(a,o)}_{D} j; x \xleftarrow{(a,o)}_{P} j$;
5: **end if**
6: **end for**
7: $WLD_{i,j}[2-15]$

Fig. 7. $SLD_{i,j}$ Program

Weak global delete After the application of a weak global delete operation on a permission (alp, dlp) for an access type (a, o) granted by i to j, the following post-conditions must satisfied: i no longer grants j the permission (alp, dlp) for access type (a, o); Permissions from the same access type (alp, dlp) granted to j by users other than i are intact; The permissions of all subjects that have been granted by j may change depending on whether other principals granted some permission for the same access type. A suitable situation to use the weak global

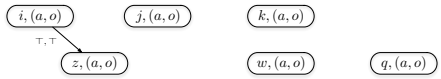

Fig. 8. Strong Global Delete

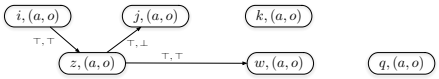

Fig. 9. Weak Global Delete

delete operation is when i loses her trust in j but she still trusts that other guarantees to make their own judgements about him. Also, since i no longer trusts j with the permission previously given, in turn she no longer trusts any subject trusted by j, and so on. In Figure 9, we show the resulting delegation chain after the execution of program $WGD_{i,j}$.

Strong global delete After the application of a strong global delete operation on a permission (alp, dlp) for an access type (a, o) granted by i to j, the following post-conditions must be satisfied: i no longer grants j the permission (alp, dlp);

$i \xleftarrow{(a,o)}_{P,D} j;$
for all $x \in \mathbb{AG}$ **do**
  if $((x,(a,o),j)^P_+ \wedge \neg\mathcal{CP}_{\{i\}}(x,(a,o)))$ **then**
    $x \xleftarrow{(a,o)}_P j; x \xleftarrow{(a,o)}_D j;$
  end if
  if $R_\emptyset(x,j,(a,o))$ **then**
    $WGD_{i,j}[2-10]$
  end if
end for

Fig. 10. $SGD_{i,j}$ *Program*

$i \xleftarrow{(a,o)}_{P,D} j;$
for all $x \in \mathbb{AG}$ **do**
  if $R_\emptyset(x,j,(a,o))$ **then**
    for all $y \in \mathbb{AG}$ **do**
      if $((y,(a,o),x)^P_+ \wedge \neg\mathcal{CP}_\emptyset(y,(a,o)))$ **then**
        $y \xleftarrow{(a,o)}_P x; y \xleftarrow{(a,o)}_D x;$
      end if
    end for
  end if
end for

Fig. 11. $WGD_{i,j}$ Program

Positive permissions for the same access type (a,o) granted to j or any descendant of j by every user z other than i are intact if they are independent of i. Otherwise, they are adjusted (i.e., restricted) to satisfy the connectivity property for those paths from z back to a SOA that is independent of i. Negative permissions of the same type are intact; The permissions of all subjects that have been granted either directly or transitively, by j may have been adjusted in order for the connectivity property to be satisfied. In Figure 8 we show the resulting delegation chain after the execution of program $SGD_{i,j}$.

Negative Revocation Schemes Negative revocation schemes differ from positive ones in that revocation is done not by deleting a positive certificate but by issuing a negative permission. The outcome of such schemes is exactly the same as the positive ones (permission is revoked) but a negative permission make it easier to go back to the previous state when negative permission is in turn revoked. For this reason we refer to [9] for an intuitive description of the schemes.[3]

6 An Epistemic Approach to Trust

The outcomes of executing a delegation or a revocation action, as presented in Sections 4 and 5, depend only on the authorization policy. The decision points of the programs presented so far are checked against the presence of information that is at system (institutional) level, like "is this agent a source of authority?" or "do we have evidence of a particular delegation certificate being held?".

However, one of the features of MAS is that agents are autonomous and therefore they can act w.r.t. a subjective and internal perception of the environment. We next show that this subjective dimension can be naturally accommodated in our logic by explicitly representing *beliefs* of agents with a standard epistemic modal operator.

A crucial subjective dimension in authorization is the one of *trust* among agents. In particular, we are interested in policy requirements like: *"An agent i trusts agent j on (a,o) while j is not considered trustworthy by agent k".*

[3] For space constraints we refer to a companion technical report [4] for a formalization of negative schemes.

The possibility of expressing subjective statements about trust enriches the model, which we describe above, in several respects:
Verification: For a verifier to grant a privilege it is not sufficient that the delegation chain is rooted according to Definition 4, but we require the chain to be such that all the agents are trusted by the verifier.
Delegation: An agent i delegates a permission to agent j not only if i has the privilege to do so but also if i trusts j.
Revocation: The introduction of trust can generalize the revocation schemes presented in Section 5. To see that, suppose that agent i wants to revoke a permission from agent j, then depending on whether i trusts j or not: 1. The agent i may want to remove the same permission from of all the other agents delegated by j that are not trusted by i; 2. The agent i may force all the other agents that gave the same privilege to j to revoke it if i does not trust them.

In what follows, we give a formal account of how to accommodate trust in all of the different respects that we reported above.

Definition 12. *A* trust model *is a tuple* $M = (W, R, V, w)$ *where:* W *is a set of possible worlds and* $w \in W$*;* $R : \mathbb{AG} \to 2^{W \times W}$ *is a function assigning to each agent an accessibility relation on* W*;* $V : \Phi \to 2^W$ *is a function assigning to each propositional letter a set of possible worlds.*

Definition 13. *The language* $\mathcal{L}_T$ *is defined inductively as follows:*

$$\mathcal{L} : \phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid B_j\phi$$

where p *in* $\Phi_T = \{t(i, (a, o)), \mid a \in \mathbb{A}, o \in \mathbb{O}, i, j \in \mathbb{AG}\}$.

The truth conditions of the relation $M, w \models \phi$ *are defined inductively as usual (we omit* $\neg$ *and* $\wedge$*).*

$$\begin{array}{lll} M, w \models p & \textit{iff} & w \in V(p) \\ M, w \models B_j\phi & \textit{iff} & \textit{for all } v \in R_j(w), M, v \models \phi \end{array}$$

Intuitively, $t(j, (a, o))$ reads as: "j is trusted on (a, o)" and $B_i t(j, (a, o))$ reads as: "i trusts j on (a, o)".

In the remainder of this section, we show how we can (independently) merge the trust model as described above, with the delegation model introduced in previous sections.

Definition 14. *A* trust-authorization model *is a pair* $\{(M, w), \Theta\}$ *of an internal trust-model* (M, w) *and a valuation* Θ *on* Φ*.*

Definition 15. *We define inductively the language* $\mathcal{L}$ *as follows:*

$$\phi ::= p \mid \psi \mid \phi \wedge \phi \mid \neg\phi \mid [\pi]\phi \qquad \pi ::= +p \mid -p \mid \phi? \mid \pi \cup \pi \mid \pi; \pi$$

where p *ranges over* Φ *and* ψ *ranges over* $\mathcal{L}_T$*. Its truth conditions on the set of internal trust delegation models are defined as follows (we omit* $\neg$ *and* $\wedge$*):*

$$\begin{array}{lll} \{(M, w), \Theta\} \models p & \textit{iff} & \Theta \models p \\ \{(M, w), \Theta\} \models \psi & \textit{iff} & M, w \models \psi \\ \{(M, w), \Theta\} \models [\pi]\phi & \textit{iff} & \{(M, w), \Theta^\pi\} \models \phi \end{array}$$

Theorem 3. *The semantics of the language $\mathcal{L}$ is completely axiomatized by the following axiom schemes and inference rules:*

L_D	*All axiom schemes and inference rules of* L_D
K_B	$\vdash B_j(\phi \to \phi') \to (B_j\phi \to B_j\phi')$
$\mathsf{Red5}$	$\vdash [\pi]\psi \leftrightarrow \psi$
Nec	*If* $\vdash \phi$ *then* $\vdash B_j\phi$

where ψ ranges over $\mathcal{L}_T$ and j over $\mathbb{AG}$.

More generally, as put forward by Abadi in [2], the use of an (epistemic) modal language permits to specify and reason about authorization in distributed environments by associating policies (i.e., formulae) to agents. For space constraints, we only give some examples of how to express such policies: If the computer science department supports that the university is trusted on (a, o), then the department will trust any other agent trusted by the university on the same access type: $(B_{cs_dep}t(uni, (a, o)) \to \bigwedge_{j \in \mathbb{AG}} (B_{uni}t(j, (a, o)) \to B_{cs_dep}t(j, (a, o))))$; If j does not trust i then he will not trust any other agent k that delegates a permission to j: $(B_j(\neg t(i, (a, o))) \to \bigwedge_{k \in \mathbb{AG}} ((k, (a, o), j)^P_+ \to B_j\neg t(k, (a, o))))$; Whatever is supported by the university is supported by the computer science department too: $(B_{uni}\varphi \to B_{cs_dep}\varphi)$, for any φ[4].

Verification When a verifier i has to check whether an agent j is permitted to perform action a on object o, she does not check for a rooted chain in which all the agents involved are trusted by i. Note that this is an inherently internal perspective which is independent from the external point of view of institutional notions, like authorization and permission. Faced with the same request, two verifiers can react differently depending on which agents they trust.

Definition 16. *In a trust-authorization system represented by $\{(M, w), \Theta\}$ a verifier i supports that j has the privilege for (a, o) iff $\{(M, w), \Theta\} \models \mathcal{DT}_\emptyset(i, j, (a, o))$, where*

$$\begin{aligned}
&\mathcal{DT}_S(i, j, (a, o)) = B_i t(j, (a, o)) \wedge \\
&\quad (\bigvee_{w \notin S} \big(((w, (a, o), j)^P_+ \wedge \neg(w, (a, o), j)^P_- \wedge soa_{w,o} \\
&\quad \wedge B_i t(w, (a, o)) \wedge B_i t(j, (a, o))) \vee \\
&\quad ((w, (a, o), j)^P_+ \wedge \neg(w, (a, o), j)^P_- \wedge B_i t(w, (a, o)) \wedge B_i t(j, (a, o)) \\
&\quad \wedge \mathcal{DT}^{P,D}_{S \cup \{w,j\}}(i, w, (a, o))\big)
\end{aligned}$$

$$\begin{aligned}
&\mathcal{DT}^{P,D}_S(i, j, (a, o)) = B_i t(j, (a, o)) \wedge \\
&\quad (\bigvee_{w \notin S} \big(((w, (a, o), j)^{P,D}_+ \wedge \neg(w, (a, o), j)^{P,D}_- \wedge soa_{w,o} \\
&\quad \wedge B_i t(w, (a, o)) \wedge B_i t(j, (a, o))) \vee \\
&\quad ((w, (a, o), j)^{P,D}_+ \wedge \neg(w, (a, o), j)^{P,D}_- \wedge B_i t(w, (a, o)) \wedge B_i t(j, (a, o)) \wedge \\
&\quad \mathcal{DT}^{P,D}_{S \cup \{w,j\}}(i, w, (a, o))\big)
\end{aligned}$$

[4] This formula has to be intended as an axiom schema, the corresponding canonical property is: $\forall x, y(xR_{cs_dep}y \to xR_{uni}y)$.

Delegation Also delegation schemes can be naturally parameterized in terms of a subjective dimension of trust. For instance, w.r.t. *delegation via transfer* we can define the following programs: - (`if` $(soa_{i,o} \wedge B_i(t(j,o)))$ `then` $+soa_{j,o}$) - (`if` $(soa_{i,o} \wedge B_i(t(j,o)))$ `then` $-soa_{i,o}; +soa_{j,o}$)

Revocation The schemes in Section 5 can be generalized with the revocation program in Figure 12 whose effects depend on the trust relationships between the revokee and the other agents in the delegation chain. The program generalizes the *weak/strong* and *global/local* dimensions of positive[5] revocation algorithms as presented in Section 5. For instance, in [9] $WGD_{i,j}$ is motivated as "...agent i loses trust in agent j but still trusts other agents to make their own judgement on j". The block $TBR[21-25]$ generalizes precisely this case, depending on whether i trusts other agents that are not independent from him, the relative permission may be revoked.

1: $i \xleftarrow{(a,o)}_{P,D} j$;
2: **if** $B_i t(j,(a,o))$ **then**
3: **for all** $x \in \mathbb{AG}$ **do**
4: **if** $(x,(a,o),j)^P_+ \wedge \neg\mathcal{CP}_{\{i\}}(x,(a,o)) \wedge B_i \neg t(x,(a,o))$ **then**
5: $x \xleftarrow{(a,o)}_{P,D} j$;
6: **end if**
7: **if** $((j,(a,o),x)^P_+ \wedge \neg\mathcal{CP}_{\emptyset}(j,(a,o)))$ **then**
8: $j \xleftarrow{(a,o)}_{P} x$;
9: **if** $(\neg\mathcal{CP}_{\emptyset}(x,(a,o)) \wedge B_i t(x,(a,o)))$ **then**
10: $i \xrightarrow{(a,o)}_{P} x$;
11: **end if**
12: **end if**
13: **if** $((j,(a,o),x)^D_+ \wedge \neg\mathcal{CP}_{\emptyset}(j,(a,o)))$ **then**
14: $j \xleftarrow{(a,o)}_{D} x$;
15: **if** $(\neg\mathcal{CP}_{\emptyset}(x,(a,o)) \wedge B_i t(x,(a,o)))$ **then**
16: $i \xrightarrow{(a,o)}_{D} x$;
17: **end if**
18: **end if**
19: **end for**
20: **end if**
21: **if** $B_i \neg t(j,(a,o))$ **then**
22: **for all** $x \in \mathbb{AG}$ **do**
23: **if** $((x,(a,o),j)^P_+ \wedge \neg\mathcal{CP}_{\{i\}}(x,(a,o)) \wedge B_i t(x,(a,o))$ **then**
24: $x \xleftarrow{(a,o)}_{P} j; x \xleftarrow{(a,o)}_{D} j$;
25: **end if**
26: **if** $R_{\emptyset}(x,j,(a,o)) \wedge \neg B_i t(x,(a,o))$ **then**
27: **for all** $y \in \mathbb{AG}$ **do**
28: **if** $((y,(a,o),x)^P_+ \wedge \neg\mathcal{CP}_i(y,(a,o)))$ **then**
29: $y \xleftarrow{(a,o)}_{P} x; y \xleftarrow{(a,o)}_{D} x$;
30: **end if**
31: **end for**
32: **end if**
33: **end for**
34: **end if**

Fig. 12. Trust Based Revocation Program $TBR_{i,j}$

7 Related Work

As we have stressed throughout our discussion, the delegation-revocation framework described by Hagström *et al.* is the basis for much of what we have described. The Hagström *et al.* work gives a semi-formal account of a range of delegation-revocation schemes, which we have formally represented in the logic language that we have introduced. We have also described an extension that allows for representing and reasoning about the beliefs.

We note that ABLP logic [3] and the RT^D model [12] allow for some restricted forms of delegation policies to be represented, but neither approach accommodates the rich range of delegation *and* revocation schemes that our approach admits. SPKI/SDSI [6] allows for delegation of privileges on objects via

[5] The algorithm can be adapted to work over negative permissions.

authorization certificates. However, the delegation policies that may be represented in the SPKI/SDSI approach are limited to a simple 1-step passing on of privileges on objects; revocation is limited to being typically effected via the expiration of short-lived certificates.

Hoek et al. [14] introduce a logic to reason how the abilities of agents and coalitions of agents are altered by transferring control from one agent to another. They adopt a dynamic propositional language in which atomic programs are of the form "agent i transfers the control of variable p to agent j". Herzig et al. [10] generalize the logic introduced in [14] by relaxing the assumption that at most one agent can control a variable. Nevertheless, delegation is still modelled as transfer and it is not possible to keep track of the delegation chain. In [13], the main focus is on reasoning about the dynamics of how responsibility can be acquired, transferred and discharged; delegation is analyzed in relation to obligations. The approach of accounting for delegation in terms of obligation creation has some merit, but the proposal does not naturally accommodate the very rich delegation-revocation framework that we have described. The work by Demolombe [7] is related to ours in the sense that an epistemic logic is described for reasoning about trust. However, Demolombe does not consider trust in the context of the range of delegation-revocation schemes that we have.

8 Conclusions and Further Work

Recall that the principal research question that we have considered is how to define a formal framework to model and reason about management structures for distributing access privileges in multi-agent systems? On that, we have described a very general framework for modelling and reasoning about delegation and revocation schemes in the context of multi-agent authorization. In particular, we introduced a (dynamic) propositional logic (Section 3) for formulating policies, we demonstrated how a range of delegation schemes (Section 4) and revocation schemes (Section 5) can be treated formally within our logic language. Our logic enables the effects of delegation and revocation actions to be expressed in terms of the changes they make to a delegation graph. The effects of performing delegation and revocation actions are expressible in terms of the "programs" that we have defined. Evidence for the applicability of our formalization is apparent in our demonstration that the eight revocation schemes informally presented in [9] and the delegation types presented in [8] can be represented in our formal framework. We also showed (Section 6) how a notion of trust can be incorporated into an extended form of our delegation-revocation framework. To the best of our knowledge, ours is the first logical framework for distributed authorization that is able to represent the range of delegation-revocation schemes that are described in [9] and [8] and that accommodates an epistemic language for explicitly representing trust relations among agents.

In terms of future work, we plan to extend the epistemic model for trust that we have introduced (Section 6). In distributed authorization, it is often quite reasonable to model trust as a simple relation between predicates (see

[16]). However, in MAS things can be more complex. For instance, we may need to admit a transitive model of trust [11] and express policies like "If i believes that j trusts z then i believes that z is trustworthy" (e.g., $B_i B_j t(z,(a,o)) \rightarrow B_i t(z,(a,o))$). In such cases, it is useful to have a modal language to nest belief modalities. The development of such a language is a matter for future work. We also intend to investigate the possibility of further developing our delegation-revocation framework to incorporate a notion of time, e.g., for time-constrained delegation of privileges on objects.

Acknowledgements Valerio Genovese is supported by the National Research Fund, Luxembourg. The authors thank the reviewers for their comments, which proved to be helpful for improving the clarity of the paper.

References

1. *FPDAM on Certificate Extensions. Final Proposed Draft Amendment on Certificate Extensions (V6), Collaborative ITU and ISO/IEC USA, Apr. 1999.*
2. M. Abadi. Logic in access control. In *18th IEEE Symposium on Logic in Computer Science (LICS)*, pages 228–, 2003.
3. M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Trans. Program. Lang. Syst.*, 15(4):706–734, 1993.
4. G. Aucher, S. Barker, G. Boella, V. Genovese, and L. van der Torre. Dynamics in delegation and revocation schemes: a logical approach (technical report), 2011. Available at `http://www.di.unito.it/~genovese/publications.html`.
5. G. Boella and L. W. N. van der Torre. Delegation of power in normative multiagent systems. In *Procs. of DEON*, volume 4048, pages 36–52, 2006.
6. D. E. Clarke, J.-E. Elien, C. M. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certificate chain discovery in SPKI/SDSI. *J. Comp. Sec.*, 9(4):285–322, 2001.
7. R. Demolombe. Reasoning about trust: A formal logical framework. In *Procs. of iTrust*, pages 291–303, 2004.
8. B. S. Firozabadi, M. J. Sergot, and O. L. Bandmann. Using authority certificates to create management structures. In *Security Protocols Workshop*, volume 2467 of *LNCS*, pages 134–145. Spinger, 2001.
9. Å. Hagström, S. Jajodia, F. Parisi-Presicce, and D. Wijesekera. Revocations-a classification. In *Procs. of CSFW-14*, pages 44–58, 2001.
10. A. Herzig and N. Troquard. The dynamic logic of propositional control. *Procs. of LIS@ESSLLI2010*, pages 107–121, 2010.
11. N. Kuntze and A. U. Schmidt. Transitive trust in mobile scenarios. In *Procs. of ETRICS*, LNCS, pages 73–85, 2006.
12. N. Li, J. Mitchell, and W. Winsborough. Design of a role-based trust-management framework. In *IEEE Symp. on Sec. and Privacy*, pages 114–130, 2002.
13. T. J. Norman and C. Reed. A logic of delegation. *Artif. Intell.*, 174(1):51–71, 2010.
14. W. van der Hoek, D. Walther, and M. Wooldridge. Reasoning about the transfer of control. *J. Artif. Intell. Res. (JAIR)*, 37:437–477, 2010.
15. C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. SPASS version 3.5. In *Procs. of CADE*, pages 140–145, 2009.
16. W. Wen and F. Mizoguchi. An authorization-based trust model for multiagent systems. *Applied Artif. Intell.*, 14(9):909–925, 2000.