UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

ABDUCTION AND DEDUCTION IN LOGIC PROGRAMMING FOR ACCESS CONTROL FOR AUTONOMIC SYSTEMS

Hristo Koshutanski and Fabio Massacci

June 2005

Technical Report # DIT-05-053

Abduction and Deduction in Logic Programming for Access Control for Autonomic Systems

Hristo Koshutanski and Fabio Massacci

Dip. di Informatica e Telecomunicazioni - Univ. di Trento
via Sommarive 14 - 38050 Povo di Trento (ITALY)

June 2005

Abstract

Autonomic communication and computing is the new paradigm for dynamic service integration over a network. An autonomic network crosses organizational and management boundaries and is provided by entities that see each other just as partners that need to collaborate with little known or even unknown parties. Policy-based network access and management already requires a paradigm shift in the access control mechanism: from identity-based access control to trust management and negotiation, but even this is not enough for cross-organizational autonomic communication. For many services no autonomic partner may guess a priori what will be sent by clients and clients may not know a priori what credentials are demanded for completing a service, which may require the orchestration of many different autonomic nodes.

To solve this problem we propose to use interactive access control: servers should be able to get back to clients asking for missing or excessing credentials, whereas the latter may decide to supply or decline requested credentials and so on until a final decision is taken.

This proposal is grounded in a formal model on policy-based access control. It identifies the formal reasoning services of deduction, abduction and consistency checking that characterize the problem. It proposes two access control algorithms for stateless and stateful autonomic services and shows their completeness and correctness.

1

Contents

1 Introduction

Recent advances of Internet technologies and globalization of peer-to-peer communications offer for organizations and individuals an open environment for rapid and dynamic resource integration. In such an environment federations of heterogeneous systems are formed with no central authority and no unified security infrastructure. Considering this level of openness each server is responsible for the management and enforcement of its own security policies with a high degree of autonomy.

Controlling access to services is a key aspect of networking and the last few years have seen the domination of policy-based access control. Indeed, the paradigm is broader than simple access control and one may speak of *policy-based network self-management* (See [36, 30] or the IEEE Policy Workshop series for examples). The intuition is that actions of nodes controlling access to services are automatically derived from policies. The nodes look at events, requests and credentials presented to them, evaluate the rules of their policies according those new facts and derive the actions [36, 37]. Policies can be "simple" `iptables` configuration rules for Linux firewalls[1] or complex logical policies expressed in languages such as Ponder [7] or a combination of policies across heterogeneous systems as in OASIS XACML framework [40].

Dynamic coalitions and autonomic communication add new challenges: a truly autonomic network is born when nodes are no longer within the boundary of a single enterprise, which could deploy its policies on each and every node and guarantee interoperability. An autonomic network is characterized by properties of self-awareness, self-management and self-configuration of its constituent nodes. In an autonomic network, nodes are partners that offer services and lightly integrate their efforts into one (hopefully coherent) network. This cross enterprise scenario poses novel security challenges with aspects of both trust management systems and workflow security.

From trust management systems [38, 10, 27] it takes the credential-based view. Since access to network services is offered by autonomic nodes on their own to potentially unknown clients, the decision to grant or deny access can only be made on the basis of the credentials sent by the client. In contrast with trust management systems, we have a continuous process and thus a notion of assignment of permissions to credentials must look beyond a single access decision.

From workflow access control systems (see e.g. [2, 4, 13, 17]) we borrow all classical problems such as dynamic assignment of roles to users, dynamic separation of duties, and assignment of permissions to users according to the least privilege principles. In contrast with workflow security management schemes, we can no longer assume that an enterprise will assign tasks and roles to users (its employees) in such a way that makes the overall flow possible w.r.t. its security constraints. The reason is that the enterprise itself no longer exists.

In an autonomic communication scenario a client might have all the necessary credentials to access a service but may simply not know it. Equally, it is unrealistic to assume that servers will publish their security policies on the web so that a client can do a policy combination and evaluation itself. So, it should be possible for a server to ask a client, on the fly, for additional credentials and the client may disclose or decline to provide them. Then the server can re-evaluate the client's request, considering the newly submitted credentials, and iterate the process until a final decision (of grant or deny) is taken. We call this modality *interactive access control*.

Part of these challenges can be solved by using policy-based self-management of networks but not all of them. Indeed, if we abstract away the details of the policy implementation, we can observe that the only reasoning service that is actually used by policy-based self-management approaches

---

[1]See `http://www.netfilter.org/`.

3

is *deduction*: given a policy and a set of additional facts find out all consequences (actions or obligations) of the policy and the facts. We simply look whether granting the request can be deduced from the policy and the current facts. Policies can be different [3, 27, 5, 4] but the kernel reasoning service is the same.

1.1 Innovation of the Framework

Access Control for autonomic communication needs another reasoning service: *abduction* [35]. Loosely speaking, we could say that abduction is deduction in reverse: given a policy and a request to access a network service, we want to know what credentials/events would grant access. Logically speaking, we want to know whether there is a (possibly minimal) set of facts that could be added to the policy so that the request can be deduced from it.

If we look again at our intuitive description of interactive access control, it is immediate to realize that abduction is the core service needed by policy-based autonomic servers. Indeed, we might even want the same service to run on both client and server sides whenever the client also requires some evidence from the server in order to establish trust before disclosing his own credentials.

Here, we present a framework for reasoning about interactive access control for autonomic communication that answers these challenges and that is grounded in a formal theory by using logic-based policies.

The intuition behind an interactive access control algorithm is the following.

- Initially a client submits a set of credentials and a service request.

- Then the algorithm checks whether the request is granted by the access policy according to the client's set of credentials.

- If the check fails then the algorithm computes all credentials disclosable from the disclosure policy according to the presented credentials.

- After that, using abductive reasoning, the algorithm finds a (minimal) solution set of missing credentials that unlocks the desired resource and preserves the access policy consistent.

- If such a set cannot be found, the algorithm performs a recovery step in which it runs the abductive reasoning again to find a (minimal) set of excessing credentials that ban the client to get a solution for the resource.

- Once a solution (missing or excessing credentials) is found it is communicated back to the client so that he can provide the missing credentials and revoked the excessing ones.

This basic procedure needs to be enhanced if we consider stateful access policies which are essentially non-monotone. For example, if a client can only access a service 3 times, clearly, presenting the same set of credentials together with the information that this is the fourth time would not lead to a grant decision. Separation of duties might be an additional reason for non-monotonicity.

In contrast to intra-enterprise workflow systems [4], a partner offering services has no way to assign to a client the right set of credentials which would be consisted with his future requests (because the partner cannot assign to or prohibit the client future tasks). So, we must have some roll-back procedure by which, if the user has by chance sent the "wrong" credentials, he can revoke them.

2   A Motivating Example

Let us assume that we have a Planet-Lab shared network between the University of Trento and Fraunhofer institute in Berlin in the context of the E-NEXT network. For the sake of simplicity let us also assume that there are three main access types to resources: *disk* – read access to data residing on the Planet-Lab machines; *run* – execute access to data and possibility to run processes on the machines; and *configure* – including the previous two types of accesses plus the possibility of configuring network services on the machines.

We also suppose that all Planet-Lab credentials (certificates) are signed and issued by trusted authorities and that the validation of these credentials is performed before the actual access control process. This can be done by plugging in any standard Privilege Management Infrastructure (PMI).

Defining the access policy, Fraunhofer institute and University of Trento decide to allow:

– `disk` access to Planet-Lab resources to anybody (any request coming) from the two institutions,

– `run` access to the network content to any member of the Planet-Lab joint hierarchy model (shown later on in Figure 2) or, if not a member, the request should come from dedicated for that internal machines of the institutions,

– `configure` access to anybody that has `run` access to the network resources and is at minimum researcher at University of Trento or junior researcher at Fraunhofer institute. Further extending access rights, `configure` access is also granted to associate professors or senior researchers with the lightened requirement of accessing the Planet-Lab network from the respective country domain of Italy or Germany. The least restrictive access is granted to full professors or members of board of directors obliging them to provide the appropriate credential for that.

Now, examine the case in which a senior researcher at Fraunhofer institute wants to have access to the system from his home place (deciding to work at home) presenting its employee certificate assuming that it is potentially enough to get read access to certain documents. But, according to the policy rules the system should deny the request because `disk`, `run` or `configure` access requests coming from domains different than University of Trento or Fraunhofer institute are allowed only to associate professors or senior researchers or higher role-positions.

So, the natural question is, "is it always the behavior we want from the system?" Shall we leave him harassing from being idle for the whole day simply because he did not know or just has forgotten that access to the system outside Fraunhofer needs another certificate?

The full formalization of the example we shall see later in §7.

3   The Basic Framework

Using Datalog and logic programs for representing and reasoning about access control is customary in computer security [3, 27, 5, 4] and this work is no exception. Our formal model for reasoning about access control is based on variants of Datalog with the stable model semantics and combines in a novel way a number of features:

• logic for trust management by Li et al. [27];

5

- logic for workflow access control by Bertino et al. [4];

- logic for disclosure and access control by Bonatti and Samarati [5];

- some ideas from trust negotiation by Yu, Winslett and Seamons [42].

We consider the view of a single partner since we cannot assume sharing of policies between partners.

In our framework each partner has a *security policy for access control* $\mathcal{P}_\mathcal{A}$ and a *security policy for disclosure control* $\mathcal{P}_\mathcal{D}$. The policy for access control is used for making decision about usage of all web services offered by a partner. The policy for disclosure control is used to decide credentials whose need can be potentially disclosed to a client. In other words, $\mathcal{P}_\mathcal{A}$ protects partner's resources by stipulating what credentials a requestor must satisfy to be authorized for a particular resource while, in contrast, $\mathcal{P}_\mathcal{D}$ defines which credentials among those occurring in $\mathcal{P}_\mathcal{A}$ are disclosable so, if needed, can be demanded from the requestor.

We also keep a set of *active credentials* $\mathcal{C}_\mathcal{P}$ that have been presented by the client in past interactions within the same service session and a set of *declined credentials* $\mathcal{C}_\mathcal{N}$ also compiled from the client's past interactions. So, to execute a service in the fragment of a partner a user will submit a set of *presented credentials* $\mathcal{C}_p$ and a *service request* r. In the same context, $\mathcal{C}_\mathcal{N}$ is computed as a difference between the *missing credentials* $\mathcal{C}_\mathcal{M}$, the client was asked in the last interaction, and the credentials presented in the current step, namely $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{M} \setminus \mathcal{C}_p$.

4   Syntax

For the syntax we build upon [4, 5, 27]. We have three disjoint sets of constants: one for users identifiers denoted by $\mathsf{User}\!:\!U$; one for roles denoted by $\mathsf{Role}\!:\!R$; and one for services denoted by $\mathsf{Service}\!:\!S$.

The predicates can be divided into three classes: predicates for assignments of users to roles and services (Fig. 1a), predicates for credentials (Fig. 1b), and predicates describing the current status of the system (Fig. 1c). The last class of predicates keeps track on the main activities done by users and services, such as: a predicate specifying successful activation of services by users; a predicate for successful completion of services; its dual one for abortion; predicates indicating granting a service to a user and, the opposite one, denial user's access to a service.

Furthermore, for some additional workflow constraints we need to have some meta-level predicates that specify how many statements are true. We use here a notation borrowed from Niemelä smodels system, but we are substantially using the count predicates defined by Das [8]:

$n \leq \{X.\, Pr\}$ where n is a positive integer, X is a set of variables, and Pr is a predicate, so that intuitively $n \leq \{X.\, Pr\}$ is true in a model if at least n instances of the grounding of X variables in Pr are satisfied by the model. The $\{X.\, Pr\} \leq n$ is the dual predicate.

We assume additional comparison predicates (for instance for equality or inequalities) or some additional monadic predicates for instance to qualify services or users.

We note here that the model, presented in the this section, can be adapted to *any* generic policy framework. The information we need from the underlying policy model is shown in Figure 1(a, b) and that information can be found in (extracted from) most policy languages.

$\mathsf{Role}{:}R_i \succ \mathsf{Role}{:}R_j$ when role $\mathsf{Role}{:}R_i$ dominates role $\mathsf{Role}{:}R_j$.

$\mathsf{Role}{:}R_i \succ_{\mathsf{Service}S} \mathsf{Role}{:}R_j$ when for service $\mathsf{Service}{:}S$, role $\mathsf{Role}{:}R_i$ dominates role $\mathsf{Role}{:}R_j$.

$\mathsf{assign}\,(P, \mathsf{Service}{:}S)$ when access to $\mathsf{Service}{:}S$ is granted to P. P can be either a $\mathsf{Role}{:}R$ or $\mathsf{User}{:}U$.

(a) Predicates for assignments to Roles and Services

$\mathsf{declaration}\,(\mathsf{User}{:}U)$ it is a statement by the $\mathsf{User}{:}U$ for its identity.

$\mathsf{credential}\,(\mathsf{User}{:}U, \mathsf{Role}{:}R)$ when $\mathsf{User}{:}U$ has a credential activating $\mathsf{Role}{:}R$.

$\mathsf{credentialTask}\,(\mathsf{User}{:}U, \mathsf{Service}{:}S)$ when $\mathsf{User}{:}U$ has the right to access $\mathsf{Service}{:}S$.

(b) Predicates for Credentials

$\mathsf{running}\,(P, \mathsf{Service}{:}S, \mathsf{Number}{:}N)$ when the Nth activation of service S is executed by P.

$\mathsf{abort}\,(P, \mathsf{Service}{:}S, \mathsf{Number}{:}N)$ if the Nth activation of service S within a workflow aborts.

$\mathsf{success}\,(P, \mathsf{Service}{:}S, \mathsf{Number}{:}N)$ if the Nth activation of service S within a workflow successfully executes.

$\mathsf{grant}\,(P, \mathsf{Service}{:}S, \mathsf{Number}{:}N)$ if the Nth request of service S has been granted

$\mathsf{deny}\,(P, \mathsf{Service}{:}S, \mathsf{Number}{:}N)$ if the Nth request of service S has been denied.

(c) Predicates describing the current status of services

Figure 1: Predicates Used in the Model

Policies are written as normal logic programs [1]. These are sets of *rules* of the form:

$$A \quad \leftarrow \quad B_1, \ldots, B_n, \; not\, C_1, \ldots, \; not\, C_m \tag{1}$$

where A, B_i and C_i are (possibly ground) predicates among those described in §4. A is called the *head* of the rule, each B_i is called a *positive literal* and each $not\, C_j$ is a *negative literal*, whereas the conjunction of the B_i and $not\, C_j$ is called the *body* of the rule. If the body is empty the rule is called a *fact*. A normal logic program is a set of rules.

In our framework, we also need *constraints* that are rules with an empty head.

$$\leftarrow \quad B_1, \ldots, B_n, \; not\, C_1, \ldots, \; not\, C_m \tag{2}$$

5 Semantics

One of the most prominent semantics for normal logic programs is the *stable model semantics* proposed by Gelfond and Lifschitz [12] (see also [1] for an introduction). The intuition is to interpret the rules of a program P as constraints on a solution set S (a set of ground atoms) for the program itself. So, if S is a set of atoms, rule (1) is a constraint on S stating that if all B_i are in S and none of C_j are in it, then A must be in S. A constraint (2) is used to rule out from the set of acceptable models situations in which B_i are true and all C_j are false (are not acceptable).

We now consider ground rules, i.e. rules where atoms do not contain variables.

Definition 5.1 *The* reduct *P^S of a ground logic program P with respect to a set of atoms S is the definite program obtained from P by deleting:*

1. *each rule that has a negative literal not C in its body with $C \in S$;*

2. *each negative literal in the bodies of the remaining rules.*

The reduct P^S is a definite logic program. Let $M(P^S) = M_{PS}$ be the semantics of the definite logic program P^S, i.e. its minimal model.

Definition 5.2 *A set of atoms S is a stable model of a normal logic program P iff $S = M(P^S)$.*

A program can have none, one or many stable models. The definition of stable models captures the two key properties of solution sets of logic programs.

1. Stable models are minimal: a proper subset of a stable model is not a stable model.

2. Stable models are grounded: each atom in a stable model has a justification in terms of the program, i.e. it is derivable from the reduct of the program with respect to the model.

Though this definition of stable models in terms of fix points is non-constructive there are constructive definitions [1] and systems [31, 26] that can cope with ground programs having tens of thousands of rules.

Logic programs with variables can be given semantics in terms of stable models.

Definition 5.3 *The stable models of a normal logic program P with variables are those of its ground instantiation P_H with respect to its Herbrand universe[2].*

If logic programs are function free, then an upper bound on the number of instantiations is rc^v, where r is the number of rules, c the number of the constants, and v the upper bound on the number of distinct variables in each rule.

Definition 5.4 (Logical Consequence and Consistency) *Let P be a logic program and L be a (positive or negative) ground literal. L is a logical consequence of P ($P \models L$) if L is true in every stable model of P. P is consistent ($P \not\models \perp$) if there is a stable model for P.*

Definition 5.5 (Security Consequence) *A request r is a security consequence of a policy P if (i) P is logically consistent and (ii) r is a logical consequence of P.*

Definition 5.6 (Abduction) *Let P be a logic program, H a set of predicates (called hypothesis, or abducibles), L a (positive or negative) ground literal, and $\prec$ a partial order (p.o.) over subsets of H. A solution of the abduction problem is a set of ground atoms E such that*

(i) $E \subseteq H$,

(ii) $P \cup E \models L$,

(iii) $P \cup E \not\models \perp$,

(iv) any set $E' \prec E$ does not satisfy all conditions above.

Traditional p.o.s are subset containment or set cardinality. Other solutions are possible with orderings over predicates.

---

[2]Essentially, we take all constants and functions appearing in the program and combine them in all possible ways. This yields the Herbrand universe. Those terms are then used to replace variables in all possible ways thus building its ground instantiation.

6 Logical Model

In this section we give formal definitions of the security policies introduced informally in §3.

Definition 6.1 (Access Policy) *An access control policy $\mathcal{P}_\mathcal{A}$ is a logic program over the predicates defined in §4 in which*

(i) no credential and no execution atom can occur in the head of a rule,

(ii) role hierarchy atoms occur as facts,

An access request r is a ground instance of an assign (User : U, Service : S) *predicate.*

In contrast to the proposal by Bertino et al. [4] for workflows, we don't need any special rule for determining which services cannot be executed and which services must be executed by a specific user or role. Logic constraints guarantee the same result.

Definition 6.2 (Disclosure Policy) *A disclosure policy $\mathcal{P}_\mathcal{D}$ is a logic program in which no role hierarchy atom and no execution atom can occur in the head of a rule.*

Example 1 (Policy Constraints) *Consider a security policy in which having a credential for the role accountant is incompatible with the assignment of any role manager, and that the execution of a service phoneCall from user billG requires that the service answer must be executed by anybody having the role $headOfStaff$. The following rules guarantees the desired behavior:*

$$\leftarrow \mathsf{credential} \, (\mathsf{User} : U, \mathsf{Role} : accountant), \mathsf{assign} \, (\mathsf{User} : U, \mathsf{Role} : manager).$$
$$\mathsf{assign} \, (\mathsf{Role} : headOfStaff, \mathsf{Service} : answer) \leftarrow$$
$$\mathsf{running} \, (\mathsf{User} : billG, \mathsf{Service} : call, \mathsf{Number} : N).$$

□

Example 2 (Access Policy and Separation of Duty Constraints) *Consider an e-stock portal where we have roles associated to services as follows: role* eSeller *– for selling shares and bonds on the floor; role* eBuyer *– for buying shares and bonds; role* eAdvisor *– used by accredited consultants to sell their advice to other customers of the portal. Then examine the case where one could send the* eAdvisor *credential to the service publishing advisories and suggest to sell shares, and at the same time the* eBuyer *credential to the service hosting bids.*
In such situations we can define separation of duty rules:

$$customer(\mathsf{eSeller}) \leftarrow .$$
$$customer(\mathsf{eBuyer}) \leftarrow .$$
$$\leftarrow \mathsf{assign} \, (\mathsf{User} : U, \mathsf{Role} : R_1), customer(R_1), \mathsf{assign} \, (\mathsf{User} : U, \mathsf{Role} : \mathsf{eAdvisor}).$$

The access control rule on reviewing selling bids is the following:

$$\mathsf{assign} \, (\mathsf{User} : U, \mathsf{Service} : S) \leftarrow \mathsf{credential} \, (\mathsf{User} : U, \mathsf{Role} : R),$$
$$\mathsf{assign} \, (\mathsf{Role} : R, \mathsf{Service} : S).$$
$$\mathsf{assign} \, (\mathsf{Role} : R, \mathsf{Service} : reviewSell) \leftarrow \mathsf{Role} : R \succ \mathsf{Role} : \mathsf{eSeller}.$$

□

As mentioned, we will use the disclosure policy $\mathcal{P}_\mathcal{D}$ to decide which missing credentials are to be asked from the client.

Example 3 (Disclosure Policy) *Considering again the access policy in Example 2. A possible (part of) the disclosure policy $\mathcal{P}_\mathcal{D}$ could be:*

$$\text{credential}\,(\mathsf{User}\!:\!U, \mathsf{Role}\!:\!eUser)\!\leftarrow\!\text{declaration}\,(\mathsf{User}\!:\!U).$$
$$\text{credential}\,(\mathsf{User}\!:\!U, \mathsf{Role}\!:\!eSeller)\!\leftarrow\!\text{credential}\,(\mathsf{User}\!:\!U, \mathsf{Role}\!:\!eUser).$$
$$\text{credential}\,(\mathsf{User}\!:\!U, \mathsf{Role}\!:\!eSellerVIP)\!\leftarrow\!\text{credential}\,(\mathsf{User}\!:\!U, \mathsf{Role}\!:\!eSeller).$$

The second rule says: to reveal the need for a eSeller *credential there should be already a credential attesting the client as a valid user (*$\mathsf{Role}\!:\!eUser$*) of the system.* ☐

So, the request $\text{assign}\,(\mathsf{User}\!:\!fm, \mathsf{Service}\!:\!reviewSell)$ together with $\text{credential}\,(\mathsf{User}\!:\!fm, \mathsf{Role}\!:\!eUser)$ and $\text{declaration}\,(\mathsf{User}\!:\!fm)$ will yield a counter request $\text{credential}\,(\mathsf{User}\!:\!fm, \mathsf{Role}\!:\!eSeller)$ specifying the need for additional privileges necessitated to get the service.

Note that the need for a credential attesting the role *eSellerVIP*, disclosed together with *eSeller*, should not be considered as a potential output by the system because the "intuition" says that *eSeller* is enough.

Remark 1 (Notion of Minimality) *The choice of the partial order has a major impact in presence of complex role hierarchies. The "intuitive" behavior of the abduction algorithm for the extraction of the* minimal *set of security credentials is not guaranteed by the straightforward interpretation of H (abducibles) as the set of credentials and by the set cardinality or set containment orderings.*

Consider the following program:

$$\mathsf{Role}\!:\!r_2 \succ \mathsf{Role}\!:\!r_1 \leftarrow .$$
$$\text{assign}\,(\mathsf{User}\!:\!U, \mathsf{Service}\!:\!ws) \leftarrow \text{credential}\,(\mathsf{User}\!:\!U, \mathsf{Role}\!:\!R), \mathsf{Role}\!:\!R \succ \mathsf{Role}\!:\!r_1.$$

Request $\text{assign}\,(\mathsf{User}\!:\!fm, \mathsf{Service}\!:\!ws)$ has two $\subseteq$-minimal solutions:

$$\{\text{credential}\,(\mathsf{User}\!:\!fm, \mathsf{Role}\!:\!r_1)\}, \{\text{credential}\,(\mathsf{User}\!:\!fm, \mathsf{Role}\!:\!r_2)\}$$

Yet, our intuition is that the first should be the minimal one.

So, we need a more sophisticated partial order rather than set cardinality or set containment. For example, we could stipulate that $E \preceq E'$ is such that for all credentials $c \in E$ there is a credential $c' \in E'$ where $c = c'$, we can revise it so that $E \prec E'$ if for $c \in E$ there is a credential $c' \in E'$ where c' is identical to c except that it contains a role R' that dominates the corresponding role R in c. This p.o. generates the "intuitive" behavior of the abduction algorithm.

An effective approximation of the above criterion is to include extra information in credentials from the hypotheses (abducibles), specifying the position of a role in the role lattice hierarchy. We called the extra information *role weight* indicating the *highest possible* position of a role in the lattice. The counting is bottom-up starting from the most bottom role(s) and reflects the number of roles the current one dominates with the direct path between them. So, if a role occurs in two different paths in the lattice, for example, once with weight 5 and once with 4 we select 5 as the role weight. Then it is easy to select those sets with lowest possible role weights – addressing in this case the least privileged principle. This is why we need the highest possible value because we can
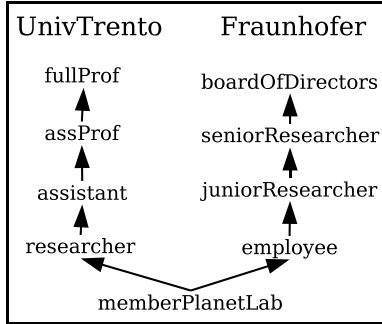
Figure 2: Joint Hierarchy Model

accurately compute the minimal privilege of the maximal importance of a role. After selecting the minimal sets (there could be more than one equally minimal) we perform a minimal set cardinality filtering to choose the final one. After having obtained the set of missing credentials, we drop this extra information from the set that is to be sent back to the client.

7 Formalization of the Example

Figure 2 shows the joint hierarchy model of the roles at both institutions, Fraunhofer and University of Trento. The partial order of roles is indicated by arcs, where higher the role in the hierarchy, more powerful it is.

Definition 7.1 *A role* dominates *another role if it is higher in the hierarchy and there is a direct path between them.*

Following is the full formalization of the running example introduced in §2. There is a preprocessing step that validates and transforms certificates to predicates suitable for the formal model – credential (User:U, Role:R). The new predicate used in the example is authNetwork ($IP, DomainName$). It is a tuple with first argument the IP address of the authorized network endpoint (the client's machine) and the second argument the domain name where the IP address comes from. Figure 3 shows the complete formalization of the policies.

Following is the functional explanation of the policies shown in Figure 3.
The access policy:

- Rules (1) and (2) give access to the shared network content to everybody from the University of Trento and Fraunhofer institute, regardless the IP and roles at these institutions.

- Rules (3) and (4) allow access from those machines that are internal for the two institutions and located in the internal LANs (dedicated machines only for Planet-Lab access) distinguished by their fixed IPs.

- Rule (5) relaxes the previous two and allows access from any place of the institutions provided users declare their ID and present some role-position certificate of their organization or at least a Planet-Lab membership credential.

- Rules (6) and (7) say that if a user has got a light access and is, at minimum, researcher at University of Trento or junior researcher at Fraunhofer, it has configure access rights.

---

    (1) assign $(*, request(disk)) \leftarrow$ authNetwork $(*, *.unitn.it)$.

    (2) assign $(*, request(disk)) \leftarrow$ authNetwork $(*, *.fraunhofer.de)$.

    (3) assign $(*, request(run)) \leftarrow$ authNetwork $(193.168.205.*, *.unitn.it)$.

    (4) assign $(*, request(run)) \leftarrow$ authNetwork $(198.162.45.*, *.fraunhofer.de)$.

    (5) assign $(User, request(run)) \leftarrow$ assign $(User, request(disk))$, declaration $(User)$,
                    credential $(User, Role)$, $Role \succeq memberPlanetLab$.

    (6) assign $(User, request(configure)) \leftarrow$ assign $(User, request(run))$, declaration $(User)$,
                    credential $(User, Role)$, $Role \succeq researcher$.

    (7) assign $(User, request(configure)) \leftarrow$ assign $(User, request(run))$, declaration $(User)$,
                    credential $(User, Role)$, $Role \succeq juniorResearcher$.

    (8) assign $(User, request(configure)) \leftarrow$ authNetwork $(*, *.it)$, declaration $(User)$,
                    credential $(User, Role)$, $Role \succeq assProf$.

    (9) assign $(User, request(configure)) \leftarrow$ authNetwork $(*, *.de)$, declaration $(User)$,
                    credential $(User, Role)$, $Role \succeq seniorResearcher$.

    (10) assign $(User, request(configure)) \leftarrow$ declaration $(User)$, credential $(User, Role)$,
                    $Role \succeq fullProf$.

    (11) assign $(User, request(configure)) \leftarrow$ declaration $(User)$, credential $(User, Role)$,
                    $Role \succeq boardOfDirectors$.

Disclosure Policy:

    (1) declaration $(User) \leftarrow$ authNetwork $(*, *.unitn.it)$.

    (2) declaration $(User) \leftarrow$ authNetwork $(*, *.fraunhofer.de)$.

    (3) credential $(memberPlanetLab, User) \leftarrow$ declaration $(User)$.

    (4) credential $(RoleX, User) \leftarrow$ credential $(RoleY, User)$, $RoleX \succ RoleY$.

Figure 3: Proxy Access and Release Policies for the Online Planet-Lab Services

- Rules (8) and (9) relaxed associate professors and senior researchers from the fact that they can access the network resources from any place they want under the respective country domain (e.g. home, other universities, etc).

- Rules (10) and (11) give full access, from any place of the world, only to members of board of directors and to full professors.

The disclosure policy:

- Rules (1) and (2) disclose the need for the client to declare its ID if the same comes from an authorized network of the respective organizations;

- Rule (3) discloses the need for Planet-Lab membership credential if the client has already declared its ID;

- Rule (4) discloses (upgrades) the need of a higher role-position credential.

8   Interactive Access Control

In this section we show how the various notions that we have seen so far can be combined in a comprehensive authorization mechanism. An authorization system receives a request r, processes

it according to the access control algorithm and eventually takes a decision. A decision may have involved interactions and so we also keep track of the current set of active credentials $\mathcal{C}_\mathcal{P}$.

Since a client must have all relevant credentials (if required) for getting access to a service, one could borrow mechanisms for certificate chain discovery [29, 6]. It is essential and important for any trust management system.

Once again it is worth noting that this view is partial as we only focus on the knowledge of one single autonomic node: there is no authorization domain crossing partnerships.

To allow for an easier grasp of the problem, we start with a basic framework shown in Figure 4. This approach is the cornerstone of most logical formalizations [9]: if the request r is a consequence of the policy and the credentials then access is granted otherwise it is denied.

---

1. verify that the request is a logical consequence of the policy ($\mathcal{P}$) and credentials ($\mathcal{C}$), namely $\mathcal{P} \cup \mathcal{C} \models r$,

2. if the check succeeds then grant access else deny access.

Figure 4: Traditional Access Control

A number of works has deemed such blunt denials unsatisfactory and therefore it has been proposed by Bonatti and Samarati [5] and Yu et al. [42] to send back to the client some of the rules that are necessary to gain additional access. Figure 5 shows the essence of the approaches. In their work it is revised to allow for the flow of rules and information to users.

---

1. verify that the request is a logical consequence of the policy ($\mathcal{P}$) and credentials ($\mathcal{C}$), namely $\mathcal{P} \cup \mathcal{C} \models r$,

2. if the check succeeds then grant access else

    (a) select some rule(s) $r \leftarrow p \in \mathtt{PartialEvaluation}(\mathcal{P} \cup \mathcal{C})$,

    (b) if such rule(s) exists then send the rule back to the client else deny access.

Figure 5: Disclosable Access Control

The systems proposed by both Bonatti and Samarati [5] and Yu et al. [42] are flat, i.e. in p the client will find all missing credentials to continue the process until r is granted. In many cases, this is neither sufficient nor desirable. For instance, if the policy is not flat, it has constraints on the credentials that can be presented at the same time (e.g., separation of duties) or a more complex role structure is used, these systems would not be complete.

Our interactive access control solution is shown in Figure 6. The intuition behind the interactive access control algorithm is the following. Initially a client will submit a set of credentials $\mathcal{C}_p$ and a service request r. $\mathcal{C}_p$ is optional and so initially may be also an empty set. Once the client has initiated a session, the interactive algorithm is started with internal input: the policy for access control $\mathcal{P}_\mathcal{A}$ and policy for disclosure control $\mathcal{P}_\mathcal{D}$.

Then the client's profile of $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{N}$ is update as: active credentials $\mathcal{C}_\mathcal{P}$ are updated with the

Global vars: $\mathcal{C}_\mathcal{N}, \mathcal{C}_\mathcal{P}, \mathcal{C}_\mathcal{M}$; Initially $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{P} = \mathcal{C}_\mathcal{M} = \emptyset$;
Input: $\mathcal{C}_p$ and r; **Internal input**: $\mathcal{P}_\mathcal{A}$ and $\mathcal{P}_\mathcal{D}$;
Output: grant/deny/ask($\mathcal{C}_\mathcal{M}$);

1. update $\mathcal{C}_\mathcal{P} = \mathcal{C}_\mathcal{P} \cup \mathcal{C}_p$,

2. update $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{N} \cup (\mathcal{C}_\mathcal{M} \setminus \mathcal{C}_p)$, where $\mathcal{C}_\mathcal{M}$ is from the last interaction,

3. verify whether the request r is a security consequence of the policy access $\mathcal{P}_\mathcal{A}$ and the active credentials $\mathcal{C}_\mathcal{P}$, namely $\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{P} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{P} \not\models \bot$,

4. <u>if</u> the check succeeds <u>then</u> return **grant** <u>else</u>

   (a) compute the set of *disclosable credentials* $\mathcal{C}_\mathcal{D}$ as
   $\mathcal{C}_\mathcal{D} = \{c \mid c \text{ credential that } \mathcal{P}_\mathcal{D} \cup \mathcal{C}_\mathcal{P} \models c\} \setminus (\mathcal{C}_\mathcal{N} \cup \mathcal{C}_\mathcal{P})$,

   (b) use abduction to find a minimal set of *missing credentials* $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$ such that both $\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M} \not\models \bot$,

   (c) <u>if</u> no such set exists <u>then</u> return **deny** <u>else</u>

   (d) return **ask($\mathcal{C}_\mathcal{M}$)** and iterate the process.

Figure 6: Interactive Access Control Algorithm

newly presented credentials $\mathcal{C}_p$; and declined credentials $\mathcal{C}_\mathcal{N}$ are updated as a set difference of what the client was asked in the last interaction ($\mathcal{C}_\mathcal{M}$) minus what he presents in the current interaction ($\mathcal{C}_p$), steps 1 and 2. After the client's profile is updated, the algorithm checks whether the request r is granted by $\mathcal{P}_\mathcal{A}$ according to the client's set of active credentials $\mathcal{C}_\mathcal{P}$ (step 3).

If the check in step 3 fails then is step 4a the algorithm computes all credentials disclosable from $\mathcal{P}_\mathcal{D}$ according to $\mathcal{C}_\mathcal{P}$ and from the resulting set removes all already declined and already presented credentials. In this case we avoid dead loops of asking something already declined or presented. Then we compute (using abduction reasoning) all possible subsets of $\mathcal{C}_\mathcal{D}$ that are consistent with the access policy $\mathcal{P}_\mathcal{A}$ and, at the same time, grant r. Out of all these sets (if any) the algorithm selects the minimal one.

Example 4 *A senior researcher at Fraunhofer institute FOKUS wants to reconfigure an online service for paper submissions of a workshop. The service is part of a big management system hosted at the University of Trento's network that is part of the Planet-Lab network described in §7. So, for doing that, at the time of access, he presents his employee membership token, issued by a Fraunhofer certificate authority, presuming that it is enough as a potential customer.*

Formally speaking, the request comes from a domain fokus.fraunhofer.de *with credential for* Role:employee *together with a declaration for a user ID,* John Milburk. *The set of credentials is:*

$$\{\text{authNetwork}\,(198.162.193.46, fokus.fraunhofer.de),$$
$$\text{credential}\,(JohnMilburk, employee),$$
$$\text{declaration}\,(JohnMilburk)\}$$

So, according to the access policy the credentials are not enough to get `configure` *access and the request would be denied (ref. rules 7,9 and 11 in Figure 3). Then, following the algorithm (step 4a in Figure 6) it is computed the set of disclosable credentials from the disclosure policy and the user's*

set of active credentials. In our case, $\mathcal{C}_{\mathcal{P}}$ is the set of credentials mentioned above. The algorithm computes $\mathcal{C}_{\mathcal{D}}$ as the need of all roles higher in position than Role:employee *(ref. Figure 3 rule 4 of Disclosure Policy):*

$$\{\text{credential}\,(\text{User}\!:\!JohnMilburk, \text{Role}\!:\!juniorResearcher),$$
$$\text{credential}\,(\text{User}\!:\!JohnMilburk, \text{Role}\!:\!seniorResearcher),$$
$$\text{credential}\,(\text{User}\!:\!JohnMilburk, \text{Role}\!:\!boardOfDirectors)\}$$

The next step, abduction (Figure 6 step 4b), computes the minimal set of credentials, out of those, that satisfies the request. The resulting set is
$\{\text{credential}\,(\text{User}\!:\!JohnMilburk, \text{Role}\!:\!juniorResearcher)\}$. *Then the need for this credential is return back to the user.*

On the next interaction step, because the user is a senior researcher, the same declines to present the requested credential and returns the same query but with no presented credentials ($\mathcal{C}_p = \emptyset$). So, the algorithm updates the user's profile marking the requested credential
$\text{credential}\,(\text{User}\!:\!JohnMilburk, \text{Role}\!:\!juniorResearcher)$ *as declined. The difference comes when the algorithm re-computes the disclosable credentials as all disclosable credentials from the last interaction minus the newly declined one:*

$$\{\text{credential}\,(\text{User}\!:\!JohnMilburk, \text{Role}\!:\!seniorResearcher),$$
$$\text{credential}\,(\text{User}\!:\!JohnMilburk, \text{Role}\!:\!boardOfDirectors)\}$$

Abduction computation returns, as a missing set, the need for credential

$$\text{credential}\,(\text{User}\!:\!JohnMilburk, \text{Role}\!:\!seniorResearcher).$$

Then, because John Milburk is a senior researcher, he presents the required certificate back to the system and gets the requested service. ◻

Remark 2 *Using declined credentials is essential to avoid loops in the process and to guarantee the success of interaction in presence of disjunctive information.*

For example suppose we have alternatives in the partner's policy (e.g., "present either a VISA or a Mastercard or an American Express card"). An arbitrary alternative can be selected by the abduction algorithm and on the next interaction step (if the client has declined the credential) the abduction algorithm is informed that the previous solution was not accepted. The process can continue until all credentials have been declined (and access is denied) or a solution is found (and access is granted).

This is all we need for autonomic processes made up by *stateless services*, in which all decisions are taken on the basis of the current input set of credentials, and which envisaged to be the large majority. This type of decision is characteristic of most logical approaches to access control [27, 4, 5]: we only look at the policy, the request and the set of credentials.

It is possible to extend the approach to stateful autonomic nodes, as we shall see in §9.

8.1 Completeness and Correctness

At first we introduce some preliminary definitions.

Definition 8.1 (Solution Set for a Resource r) *Let $\mathcal{P}_\mathcal{A}$ is an access policy and r be a request. A set of credentials $\mathcal{C}_\mathcal{S}$ is a solution set for r according to $\mathcal{P}_\mathcal{A}$ if r is a security consequence of $\mathcal{P}_\mathcal{A}$ and $\mathcal{C}_\mathcal{S}$ ($\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{S} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{S} \not\models \bot$).*

Definition 8.2 (Disclosable and Hidden Credentials) *Let $\mathcal{P}_\mathcal{D}$ be a disclosure policy, credential c is* disclosable *if there is a set of credentials $\mathcal{C}$ s.t. $c \notin \mathcal{C}$ and $\mathcal{C}$ together with the disclosure policy $\mathcal{P}_\mathcal{D}$ entails c, namely $\mathcal{P}_\mathcal{D} \cup \mathcal{C} \models c$.*
A credential c is hidden *if it is not disclosable.*

The intuition behind hidden credentials is that the system *does not ask* for them but *expects* them from the client. So, the information for hidden credentials is obtained by out-of-band sources. Also, from the point of view of a client, hidden credentials are just credentials that someone has told him to provide them when requests a specific service. Hidden credentials are used either to unlock more credentials needed to grant access or used directly to unlock a resource or used for both. So, a client must provide them when initially requests a service. Essentially, the client can work in *pull* mode with disclosable credentials and must work in *push* mode with hidden credentials.

Definition 8.3 (Hidden Credentials for a Resource r) *A set of hidden credentials for a resource r is the set $\mathcal{C}_\mathcal{H}$ such that:*

1. *there exists a solution set $\mathcal{C}_\mathcal{S}$ for r that $\mathcal{C}_\mathcal{S} \supseteq \mathcal{C}_\mathcal{H}$ and*

2. *all hidden credentials in $\mathcal{C}_\mathcal{S}$ are in $\mathcal{C}_\mathcal{H}$.*

In particular, all solution sets for a resource r could be hidden, i.e. for any solution $\mathcal{C}_\mathcal{S}$ and its set of hidden credentials $\mathcal{C}_\mathcal{H}$ holds $\mathcal{C}_\mathcal{H} = \mathcal{C}_\mathcal{S}$, and we fall back in the standard, classical, access control framework of having only grant/deny decisions. On the other hand, every solution $\mathcal{C}_\mathcal{S}$ for r with hidden credentials equal to an empty set is just a solution for r, or can be interpreted as any solution is a solution with hidden credentials where hidden credentials might be equal to empty set. Also it might be a case that some solution sets for r have the same sets of hidden credentials.

Definition 8.4 (Fair Access) *Let $\mathcal{P}_\mathcal{A}$ be an access control policy and let $\mathcal{C}_{\mathcal{P}_\mathcal{A}}$ be the set of ground instances of all credentials occurring in $\mathcal{P}_\mathcal{A}$. The policy $\mathcal{P}_\mathcal{A}$ guarantees fair access if for any request r there exists a set $\mathcal{C}_\mathcal{S} \subseteq \mathcal{C}_{\mathcal{P}_\mathcal{A}}$ that is a solution for r.*

We can check fair access property by simply running the interactive algorithm for all resources $r \in \mathcal{P}_\mathcal{A}$ and set up the disclosable credentials to all credentials occurring in $\mathcal{P}_\mathcal{A}$. Then for each request r the algorithm should return $\mathtt{ask}(\mathcal{C}_\mathcal{M})$ for some $\mathcal{C}_\mathcal{M}$. If for some r it returns *deny* then the fair access property fails.

Definition 8.5 (Fair Interaction) *Let $\mathcal{P}_\mathcal{A}$ and $\mathcal{P}_\mathcal{D}$ be, respectively, an access and disclosure control policies. The policies* guarantee fair interaction *if*

1. *$\mathcal{P}_\mathcal{A}$ guarantees fair access and*

2. *if $\mathcal{C}_\mathcal{S}$ is a solution for a request r and $\mathcal{C}_\mathcal{H}$ is the set of hidden credentials for $\mathcal{C}_\mathcal{S}$ then the visible part of $\mathcal{C}_\mathcal{S}$ is disclosable by $\mathcal{P}_\mathcal{D} \cup \mathcal{C}_\mathcal{H}$, i.e. $\forall c \in (\mathcal{C}_\mathcal{S} \setminus \mathcal{C}_\mathcal{H})$, $\mathcal{P}_\mathcal{D} \cup \mathcal{C}_\mathcal{H} \models c$.*

The intuition of fair interaction is that the hidden credentials in a solution set are all that is needed to obtain the disclosure of the remaining disclosable credentials. For example setting up an e-mail account `my_name@google.com` is an example of fair interaction with hidden credentials. One needs a form that is not available on the site but after the form, duly filled, has been sent some additional information (credentials) is asked and the account is granted.

The intuition behind unfair interaction policies is that, even if we have all credentials necessary to access, even if we know that some credentials (the hidden ones) must be sent in push mode, yet this will not be enough to get an answer from the server. We will need to push other credentials that are not needed for access but just to disclose the information on missing credentials.

Assuming that for each request $r \in \mathcal{P}_{\mathcal{A}}$ a service provider knows a priori all sets of hidden credentials $\mathcal{C}_{\mathcal{H}1}, \ldots, \mathcal{C}_{\mathcal{H}n}$ we can check the fair interaction property by running the interactive algorithm n-times with input: r and $\mathcal{C}_{\mathcal{H}i}$, $i = [1..n]$. Then for each request r and each run the algorithm should return either $\mathtt{ask}(\mathcal{C}_{\mathcal{M}})$ or $grant$. If for some r in some runs it returns $deny$ then the fair interaction property fails.

If a service provider does not know the set of hidden credentials these can again be calculated from the disclosure policies by using the abduction algorithm on the disclosure policy.

Definition 8.6 (Powerful Client) *A powerful client is a client that whenever receives* $\mathtt{ask}(\mathcal{C}_{\mathcal{M}})$ *returns* $\mathcal{C}_{\mathcal{M}}$.

Definition 8.7 (Cooperative Client) *A client with a set of credentials* $\mathcal{C}$ *is a* cooperative client *if whenever receives* $\mathtt{ask}(\mathcal{C}_{\mathcal{M}})$ *returns* $\mathcal{C}_{\mathcal{M}} \cap \mathcal{C}$.

Definition 8.8 (Client with Hidden Credentials for a Resource r) *A client with hidden credentials for a resource r is any client that has the set of hidden credentials* $\mathcal{C}_{\mathcal{H}}$ *of a solution set for r and whenever requests r he sends* $\mathcal{C}_{\mathcal{H}}$ *initially.*

Definition 8.9 (Monotonic and Non-monotonic Policy) *A policy P is* monotonic *if whenever a set of statements $\mathcal{C}$ is a solution set for r according to P ($P \cup \mathcal{C} \models r$) then any superset $\mathcal{C}' \supset \mathcal{C}$ is also a solution set for r according to P ($P \cup \mathcal{C}' \models r$).*

In contrast, a non-monotonic *policy is a logic program in which if $\mathcal{C}$ is a solution for r it may exists $\mathcal{C}' \supset \mathcal{C}$ that is not a solution for r, i.e. $P \cup \mathcal{C}' \not\models r$*

Definition 8.10 (Resource r Additive Policy) *A policy P is* a resource r additive *if for every two sets of statements $\mathcal{C}$ and $\mathcal{C}'$, where $\mathcal{C} \not\subset \mathcal{C}'$ and $\mathcal{C}' \not\subset \mathcal{C}$, that unlock the resource r according to P then also $\mathcal{C} \cup \mathcal{C}'$ unlocks r according to P.*

In other words, if you have two solutions for a service you can "add" them and you will still get the service.

Definition 8.11 (Resource r Subset Consistent Policy) *A policy P is a* resource r subset consistent *if for every solution set $\mathcal{C}_{\mathcal{S}}$ for r holds that each $\mathcal{C} \subseteq \mathcal{C}_{\mathcal{S}}$ preserves consistency in P, i.e. $P \cup \mathcal{C} \not\models \bot$.*

The intuition behind a subset consistent policy is that inconsistency occurs because of separation of duty. So this type of constraints rule out situation in which a client has too many privileges. Situations where having less credentials than enough to get a service makes the system inconsistent are neither practical nor intuitive from an access control point of view.

Proposition 8.1 (Sufficient Condition for Subset Consistency)   *Let $\mathcal{P}_\mathcal{A}$ be an access policy. If the number of negations from a literal in the body of a constraint to a credential in $\mathcal{P}_\mathcal{A}$ is even then $\mathcal{P}_\mathcal{A}$ is subset consistent.*

Proof.   Let assume that the number of negations from a literal in the body of a rule to a credential in $\mathcal{P}_\mathcal{A}$ is even and the policy $\mathcal{P}_\mathcal{A}$ is not subset consistent. Then let $\mathcal{C}_\mathcal{S}$ be a solution set for a request r and let $\mathcal{C} \subseteq \mathcal{C}_\mathcal{S}$ such that $\mathcal{P}_\mathcal{A} \cup \mathcal{C} \models \bot$. Then because $\mathcal{C}_\mathcal{S}$ preserves consistency and the fact that no credential occurs in a head of a rule in $\mathcal{P}_\mathcal{A}$ (ref. Def. 6.1) follows that inconsistency occurs from reducing the number of credentials in the system. Reducing credentials either directly affects a constraint having negation of a literal that is a credential belonging to $\mathcal{C}_\mathcal{S} \setminus \mathcal{C}$ or indirectly affects a constraint that has a positive literal, which by its side is deduced from a rule in $\mathcal{P}_\mathcal{A}$ that has either negation or a positive literal in the body. If negation in the latter case then it is a missing credential. In both cases the number of negations is 1 which is odd. If we apply recursively the same rule for a positive literal in the latter case we will compute again odd number of negations. In any way we have a contradiction with what we assumed. With this we finish the proof.   □

Definition 8.12 (Well-behaved Policy) *A policy P is* well-behaved *if for all resources $r \in P$*

(i) *P is* resource r additive *and*

(ii) *P is* resource r subset consistent.

The set of well-behaved policies resides between monotonic and arbitrary policies.

Proposition 8.2 *All monotonic policies are well-behaved but the converse is not true.*

Proof.   In one direction the property is immediate: if a policy $\mathcal{P}_\mathcal{A}$ is monotonic then, according Def. 8.9, if a set $\mathcal{C}_\mathcal{S}$ is a solution for a request r then also any superset is. So if $\mathcal{C}_\mathcal{S}'$ is as another solution set for r, so also $\mathcal{C}_\mathcal{S} \cup \mathcal{C}_\mathcal{S}'$ is a solution because of the superset property. The subset consistency property always holds for monotonic policies since inconsistent states are not considered for those.
    For the other way round we show a counter-example:

$$
\begin{aligned}
r_1 &\leftarrow C_A.\\
r_1 &\leftarrow C_B.\\
r_2 &\leftarrow C_C.\\
&\leftarrow C_A, C_C.\\
&\leftarrow C_B, C_C.
\end{aligned}
$$

In our case having $\{C_A, C_B, C_C\}$ bans the client to get either of the services, which clearly shows that the example is a non-monotonic policy. At the same time, for each of the services we have additive and subset consistent properties so that the policy is well-behaved.   □

Remark 3 *Hereinafter all access policies $\mathcal{P}_\mathcal{A}$ will be* well-behaved *policies and all disclosure policies $\mathcal{P}_\mathcal{D}$ will be* monotonic *policies unless explicitly specified otherwise.*

Also we assume that a client initiates a service request with an empty set of presented credentials or, if hidden credentials needed, the presented credentials are the hidden ones. The assumption is important to avoid initial inconsistency and to assure that a client has a successful first step.

Whenever hidden credentials are not explicitly linked in the theorems we assume that for each request r in $\mathcal{P}_\mathcal{A}$ its set of hidden credentials $\mathcal{C}_\mathcal{H} = \emptyset$. The assumption mainly reflects the use of Definition 8.5 for fair interaction. We apply the same definition but with no $\mathcal{C}_\mathcal{H}$ meaning that for any request r each of its solution sets is disclosable by the disclosure policy.

Definition 8.13 (Completeness) *If a client has a solution for a request r then he eventually gets* grant r.

Definition 8.14 (Soundness) *If a client gets* grant r *then he has a solution for r.*

Theorem 8.1 (Soundness) *Let $\mathcal{P}_\mathcal{A}$ be an access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy and r a request. If a client gets* grant r *with the algorithm in Figure 6 then he has a solution set $\mathcal{C}_\mathcal{S}$ that unlocks r according to $\mathcal{P}_\mathcal{A}$.*

Proof. This proof is rather straightforward. Let suppose that the client, requested service r, got *grant*. The only way to get it is when $\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{P} \models r$. There are two cases: either $\mathcal{C}_\mathcal{P} = \emptyset$ or $\mathcal{C}_\mathcal{P} \neq \emptyset$.

If $\mathcal{C}_\mathcal{P} = \emptyset$ then the resource r is not protected by $\mathcal{P}_\mathcal{A}$, i.e. $\mathcal{P}_\mathcal{A} \models r$. So, the $\emptyset$ is a solution for r and the client has it.

If $\mathcal{C}_\mathcal{P} \neq \emptyset$ then the only way to introduce a credential in $\mathcal{C}_\mathcal{P}$ is by step 1 of the algorithm. Since initially $\mathcal{C}_\mathcal{P} = \emptyset$ so the client has sent a sequence of sets of credentials $\mathcal{C}_{p_1}, \ldots, \mathcal{C}_{p_n}$ such that $\bigcup_{i=1}^n \mathcal{C}_{p_i} = \mathcal{C}_\mathcal{P}$. Then the client has a set of credentials that unlocks it. $\square$

Theorem 8.2 (Termination) *Let $\mathcal{P}_\mathcal{A}$ be an access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy and r a request. The access control algorithm in Figure 6 always terminates.*

Proof. To prove this claim simply observe that at each interaction the union of the presented credentials or the declined credentials occurring in the access policy always increases. Since this set is bounded by the credentials occurring in the access policy there is always a stage in which either grant is given (enough presented credentials to unlock the service) or deny is sent (too many declined credentials to find another set of missing credentials). $\square$

8.2 Completeness for Powerful and Cooperative Clients

The most important thing is also the most difficult to prove: a client who has the right set of credentials and who is willing to send them to the server, will not be left stranded in our autonomic network and will eventually get grant.

We prove this result in stages: first for powerful clients and then for cooperative clients. We notice that it is fairly difficult to prove any results for non-cooperative clients: if they are unwilling to send the credentials to the server, how can ever the server grants them access? However, if at least the client is willing to give his credentials if the server guess the right combination can also be captured.

Theorem 8.3 (Completeness for a Powerful Client) *Let $\mathcal{P}_\mathcal{A}$ be a* non-monotonic *access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy and r a request. If $\mathcal{P}_\mathcal{A}$ and $\mathcal{P}_\mathcal{D}$ guarantee fair access and interaction then a powerful client always gets* grant r *with the algorithm in Figure 6.*

Proof. A powerful client requests r with an initial set of presented credentials equal to empty set. Then the algorithm runs steps 1 and 2. If step 3 succeeds (no credentials are needed for r) then the algorithm returns grant at step 4 and we are done.

If step 3 does not succeed then the algorithm goes to step 4a. At this point $\mathcal{C_N} = \mathcal{C_P} = \emptyset$. In this case $\mathcal{C_D}$ consists of all credentials disclosable by $\mathcal{P_D}$. Then in step 4b the abduction algorithm will return a set $\mathcal{C_M}$ that unlocks r because

(i) $\mathcal{P_A}$ guarantees fair access and so a solution set $\mathcal{C_S}$ for r exists,

(ii) since $\mathcal{P_D}$ satisfies the property fair interaction and $\mathcal{C_H} = \emptyset$ so $\mathcal{C_S}$ is disclosed by $\mathcal{P_D}$,

(iii) clearly $\mathcal{C_S}$ is a subset of $\mathcal{C_D}$, because $\mathcal{C_D}$ consists of all credentials disclosable by $\mathcal{P_D}$.

So, step 4c is not reached and in step 4d the algorithm returns $\mathtt{ask}(\mathcal{C_M})$ which satisfies the two conditions of step 4b.

If there is only one solution that unlocks r then $\mathcal{C_M} = \mathcal{C_S}$.

Since the client is a powerful client then on the next interaction he returns $\mathcal{C_M}$. Then the algorithm updates $\mathcal{C_P} = \mathcal{C_M}$ and $\mathcal{C_N} = \emptyset$ and because $\mathcal{C_M}$ satisfies the two conditions in step 4b, in the last interaction, so it also satisfies the conditions in step 3 and in step 4 it returns *grant*. $\square$

Theorem 8.4 (Completeness for a Cooperative Client) *Let $\mathcal{P_A}$ be an access policy, $\mathcal{P_D}$ be a disclosure policy and r a request. If $\mathcal{P_A}$ and $\mathcal{P_D}$ guarantee fair access and interaction then if a cooperative client has a set of credentials $\mathcal{C_S}$ that unlocks r according to $\mathcal{P_A}$ then the client always gets grant r with the algorithm in Figure 6.*

Proof. We will proof it in two steps. First step, by induction, showing that in a single interaction if a cooperative client does not get grant r then he gets $\mathtt{ask}(\mathcal{C_M})$. In other words, he will never receive denial by the algorithm. Second step, we will show that if the first step is true then a cooperative client with a solution set $\mathcal{C_S}$ always gets grant r.

Step 1.
Proof by induction on the interaction steps:

Inter. 1: Client requests service r together with an initial set of credentials $\mathcal{C}_p = \emptyset$ and we fall back exactly in the proof of Theorem 8.3 for that interaction step.

Inter. N: Here we use the induction hypothesis that the client fails to get grant r and gets $\mathtt{ask}(\mathcal{C_M})$ in the previous interaction. Now, suppose that the client fails to get grant r in step 3. The only way it fails is that there is no solution set in $\mathcal{C_P}$. It is because in $\mathcal{C_P}$ there are only credentials partially compiled from solutions for r, i.e. $\mathcal{C_P} \subset (\mathcal{C}_{\mathcal{M}1} \cup \ldots \cup \mathcal{C}_{\mathcal{M}n})$. Then since $\mathcal{P_A}$ is *well-behaved* so $\mathcal{C_P}$ preserves consistency in $\mathcal{P_A}$.

So, after the check failed in step 3 the algorithm computes the set of disclosable credentials $\mathcal{C_D}$ as all credentials disclosable by $\mathcal{P_D} \cup \mathcal{C_P}$ minus all already declined and presented credentials.

Because $\mathcal{P_A}$ and $\mathcal{P_D}$ guarantee fair access and interaction so $\mathcal{C_S} \subseteq (\mathcal{C_D} \cup \mathcal{C_P})$ but $\mathcal{C_S} \nsubseteq \mathcal{C_P}$. Then at least the set difference $\mathcal{C_S} \setminus \mathcal{C_P}$ will be computed by the abduction engine because: (i) $(\mathcal{C_S} \setminus \mathcal{C_P}) \subseteq \mathcal{C_D}$ and (ii) $(\mathcal{C_S} \cup \mathcal{C_P}) \subset (\mathcal{C}_{\mathcal{M}1} \cup \ldots \cup \mathcal{C}_{\mathcal{M}n} \cup \mathcal{C_S})$ preserves consistency (ref. Def. 8.12 and Def. 8.11). And so the step 4c is skipped the client gets $\mathtt{ask}(\mathcal{C_M})$.

Step 2. So, in Step 1 we proved that if a cooperative client does not get grant r, in a single interaction step, he gets $\mathtt{ask}(\mathcal{C_M})$. Then we have to prove that in a finite number of steps a cooperative client will always get grant r.

There is a finite number of solutions for each request r simply because $\mathcal{P_A}$ consists of a finite number of statements. The abduction reasoning service at each interaction computes different solution wrt the solutions computed in previous interactions because from the disclosable credentials we remove all those credentials from the previous interactions (ref. step 4a in Fig. 6).

So, since there are finite solution sets for r and since the client has one of them therefore according to what we proved in **Step 1** the client in a finite number of interaction steps will disclose $\mathcal{C_S}$, i.e. $\mathcal{C_S} \subseteq \mathcal{C_P}$, and get grant r. □

Theorem 8.5 (Completeness for a Powerful Client with Hidden Credentials) *Let $\mathcal{P_A}$ be an access policy, $\mathcal{P_D}$ be a disclosure policy and r a request. If $\mathcal{P_A}$ and $\mathcal{P_D}$ guarantee fair access and interaction then a powerful client with hidden credentials $\mathcal{C_H}$ for r always gets grant r with the algorithm in Figure 6.*

Proof. A powerful client requests r with an initial set of presented credentials equal to the set of hidden credentials, i.e. $\mathcal{C}_p = \mathcal{C_H}$. Then the algorithm runs steps 1 and 2. At this point $\mathcal{C_P} = \mathcal{C_H}$ and $\mathcal{C_N} = \emptyset$. If step 3 succeeds, $\mathcal{C_H}$ itself is a solution set for r, then the algorithm returns grant at step 4 and we are done.

If step 3 does not succeed then the algorithm goes to step 4a. In this case $\mathcal{C_D}$ consists of all credentials disclosable by $\mathcal{P_D}$ and $\mathcal{C_H}$ because $\mathcal{C_P} = \mathcal{C_H}$. Then in step 4b the abduction algorithm will return a set $\mathcal{C_M}$ because at least one such set exists:

(i) $\mathcal{P_A}$ and $\mathcal{P_D}$ guarantee fair access and interaction and so for the solution set $\mathcal{C_S}$, corresponding to the set of hidden credentials $\mathcal{C_H}$ that the client has, its visible part $\mathcal{C_S} \setminus \mathcal{C_H}$ will be disclosed by $\mathcal{P_D}$ and $\mathcal{C_H}$ (ref. Def. 8.5),

(ii) clearly $(\mathcal{C_S} \setminus \mathcal{C_H}) \subseteq \mathcal{C_D}$, because $\mathcal{C_D}$ consists of all credentials disclosable by $\mathcal{P_D} \cup \mathcal{C_H}$,

(iii) according to Definition 8.3 the set $\mathcal{C_H} \subseteq \mathcal{C_S}$ and so the set $\mathcal{C_S} \setminus \mathcal{C_H}$ together with $\mathcal{C_P}$ preserve consistency in $\mathcal{P_A}$ (simply because $(\mathcal{C_S} \setminus \mathcal{C_H}) \cup \mathcal{C_P} = \mathcal{C_S}$ is a solution set),

So, step 4c is not reached and in step 4d the algorithm returns $\mathtt{ask}(\mathcal{C_M})$ which satisfies the two conditions of step 4b.

It may be the case in which the set $\mathcal{C_H}$ unlocks more than one solution sets if those solution sets have the same set of hidden credentials.

Since the client is a powerful client then on the next interaction he returns $\mathcal{C_M}$. Then the algorithm updates $\mathcal{C_P} = \mathcal{C_H} \cup \mathcal{C_M}$ and $\mathcal{C_N} = \emptyset$ and because $\mathcal{C_M}$ satisfies the two conditions in step 4b, from the last interaction, so it also satisfies the conditions in step 3 and in step 4 it returns *grant*. With this we finish the proof. □

Theorem 8.6 (Completeness for a Cooperative Client with Hidden Credentials) *Let $\mathcal{P_A}$ be an access policy, $\mathcal{P_D}$ be a disclosure policy and r a request. If $\mathcal{P_A}$ and $\mathcal{P_D}$ guarantee fair access and interaction then if a cooperative client with hidden credentials $\mathcal{C_H}$ for r has a solution set $\mathcal{C_S}$ for $\mathcal{C_H}$ then the client always gets grant r with the algorithm in Figure 6.*

Proof.   Analogously of theorem 8.4 we will proof it in two steps. First step, by induction, showing that in a single interaction if the client does not get grant r then he gets $\texttt{ask}(\mathcal{C_M})$. In other words, he will never receive denial by the algorithm. Second step, we will show that if the first step is true then the client with a solution set $\mathcal{C_S}$ always gets grant r.

Proof by induction on the interaction steps:

> **Inter. 1**: Client requests service r together with an initial set of credentials $\mathcal{C}_p = \mathcal{C_H}$ and we fall back exactly in the proof of Theorem 8.5 for that interaction step.

> **Inter. N**: Here we use the induction hypothesis that the client fails to get grant r and gets $\texttt{ask}(\mathcal{C_M})$ in the previous interaction. Now, suppose that the client fails to get grant r in step 3. Then the only difference with the respective interaction step in the proof of Theorem 8.4 is that the client's solution set $\mathcal{C_S}$ is still disclosable by $\mathcal{P_D}$ and $\mathcal{C_P}$ because $\mathcal{C_H}$ is already in the set $\mathcal{C_P}$, see Definition 8.3. So, according to fair interaction property follows that $\mathcal{C_S} \setminus \mathcal{C_H}$ is visible and the abduction service at least finds it as a solution.

The rest of the proof can be done along the same line as in Theorem 8.4.   □

9   Stateful Systems

Stateful systems are systems where the status of the current state depends on the status of the system in past conditions, i.e. access decision can change depending on past interactions or past presented credentials. Such systems can easily become inconsistent wrt a client's set of presented credentials mainly because access decisions depend also on the history of past executions. To address the problem of inconsistency, this chapter extends the basic framework, described in §8, so that it allows a service provider to reason of not only what missing credentials are needed but also to find out what the excessing (conflicting) is among the clients set of presented credentials that makes the policy state inconsistent.

Past requests or services may deny access to future services as in Bertino et al [4] centralized access control model for workflows. Separation of duties means that we cannot extend privileges by supplying more credentials. For instance a branch manager of a bank clearing a cheque cannot be the same member of staff who has emitted the cheque [4, pag.67]. If we have no memory of past credentials then it is impossible to enforce any security policy for separation of duties on the application workflow. The problems that could cause a process to get stuck are the following:

- the request may be inconsistent with some role, action or event from the client in the past;

- the new set of credential may be inconsistent with requirements such as separation of duties;

To execute a service of the fragment of a partner, the user will submit a set of *presented credentials* $\mathcal{C}_p$, a set of *revoked credentials* $\mathcal{C}_r$ and a *service request* r. We assume that $\mathcal{C}_p$ and $\mathcal{C}_r$ are disjoint. We also need to keep a memory of past credentials submitted by a user. This is the role of $\mathcal{C_P}$, the set of *active credentials* that have been presented by the client in past requests to other services within the domain of a partner.

In many workflow authorization schemes, the policy alone is not sufficient to make an access control decision and thus we need to identify a *history of execution* $\mathcal{H}$ of services under the control of a partner. It keeps track on what has been done by the system and what is the current status of it.

```
Global Vars:  $\mathcal{C_P}$, $\mathcal{H}$;
Initially: $\mathcal{C_P} = \emptyset$ and $\mathcal{H} = \emptyset$;

ServiceRequest($r$, $\mathcal{C}_p$, $\mathcal{C}_r$){ // starts a new thread
    1. $result_{access}$ = InteractiveAccessControl($r$, $\mathcal{C}_p$, $\mathcal{C}_r$);
    2. Update($\mathcal{H}$, $result_{access}$, $r$);
    3. if $result_{access} == grant$ then
    4.     $result_{service}$ = InvokeService($r$);
    5.     Update($\mathcal{H}$, $result_{service}$, $r$);
    6. endif
}
```

Figure 7: Global Initialization and Service Management

9.1 Initialization and Service

Figure 7 gives the intuition of possible management of services wrt the access control decision process and its relevant data sets. The algorithm shown in Figure 7 works like web servers. A main web server listens to service requests and whenever a request is detected the server runs the algorithm in a new thread and initializes the global variables $\mathcal{H}$ and $\mathcal{C_P}$ to empty sets. Of course here it should distinguish between the very first initialization and any further loading of those data sets simply because we do not want to loose any data from past interactions. For further loadings we keep a profile for each client with the respective data set $\mathcal{C_P}$ so that when the client accesses again a service we just load $\mathcal{C_P}$.

Both $\mathcal{C_P}$ and $\mathcal{H}$ are local to a service provider and are managed and initialized independently. The history of execution $\mathcal{H}$ is set up to an empty set at the start when a particular business process is started[3]. Even a business partner may decide to have for each running business process separate histories $\mathcal{H}$. To this extend we assume that $\mathcal{H}$ is mapped to those business process(es) that are relevant to the authorization logic and is released when those processes complete their executions.

Another session profile is the set of active user's access rights $\mathcal{C_P}$ available to the partner's application domain. Each session is associated with a single user and each user is associated with a *single* session. This session profile is created when the user for the first time requests a service under the partner's domain. In contrast to the history profile, the set of user's access rights is valid until a certain time slot expires[4]. Even more, it is valid across multiple runnings of business processes within the entire scope of the partner's domain and it eventual deactivation depends on the partner's authorization logic. The set of active credentials $\mathcal{C_P}$, as shown in Figure 7, is updated as a side-effect of the execution of InteractiveAccessControl function and later, in §9.2, we will show how $\mathcal{C_P}$ is managed.

The third session profile, kept in the model, is for the service level negotiation. Here, each session is associated with a single user but each user is associated with *one or more* negotiation sessions. Whenever a user requests a service (ref. ServiceRequest(...) in Figure 7) it is created a session within which the interactive access control algorithm is running. Once the session is created the user interacts with the system until a final decision of grant or deny is taken. Within these

---

[3]It entirely depends on the provider's business logic and whenever an application business process is started we refer to it as an initial point to set up $\mathcal{H} = \emptyset$

[4]The set $\mathcal{C_P}$ is strictly time-dependent and must be periodically cleared up from already expired credentials.

interactions a user (de)activates some subset of roles ($\subseteq \mathcal{C_P}$) that he or she is assigned.

In comparison with RBAC model, the service level agreement session corresponds to the *user_sessions(u:USE* function as introduced by Ferraiolo et al. [11], which is the mapping of a user u onto a set of active sessions. The user session of active rights corresponds to *avail_session_perms(user_sessions(u:USERS))* introduced in [11]. Where *avail_session_perms(s:SESSIONS)* returns the permissions available to a user in a session. We note that the user's session of active access rights in our framework extends the one in [11] because in $\mathcal{C_P}$ one can also find access credentials from already concluded service level sessions but with still valid expiration dates.

To keep the history of execution $\mathcal{H}$ up-to-date, after each interaction step appropriate predicates should be added indicating what has been done by the system. This is done by the function Update shown in Figure 7. The table below summarizes the possible updates of $\mathcal{H}$.

Algorithm output	Status Predicates
grant	grant (User:U, Service:S, Number:N),
	running (User:U, Service:S, Number:N)
deny	deny (User:U, Service:S, Number:N)
Service Execution	Status Predicates
accomplished	success (User:U, Service:S, Number:N)
failed	abort (User:U, Service:S, Number:N)

The temporal evolution of the access rights wrt the history of execution $\mathcal{H}$ can be complex because even the most simple constraint on executed actions may block a request. Indeed, the set of requests that must be grantable by the policy may change with the services that we have used. As intuitively expected, we may have access to less services if we have limitations on their usage. For instance the following constraint specifies that the service *reviewSellBids* cannot be executed more than three times in a workflow session:

$$\leftarrow \mathsf{assign}\,(\mathsf{User}{:}\,U, \mathsf{Service}{:}\,reviewSellBids),$$
$$4 \leq \{N.\,\mathsf{success}\,(\mathsf{User}{:}\,U, \mathsf{Service}{:}\,reviewSellBids, \mathsf{Number}{:}\,N)\}\,.$$

9.2 Interactive Access Control for Stateful Systems

Once a client makes a service request, the authorization mechanism starts a session in which the client iterates with the system until a final decision of *grant* or *deny* is taken. In the same session context, we keep a set of *declined credentials* $\mathcal{C_N}$, a set of *missing credentials* $\mathcal{C_M}$ and a set of *excessing credentials*[5] $\mathcal{C_E}$. The set $\mathcal{C_N}$ consists of credentials that the client has refused to present to the system during an authorization session. The sets $\mathcal{C_M}$ and $\mathcal{C_E}$ keep information from the output of the last interaction. Once the session is started, the algorithm loads the policies for access and disclosure control $\mathcal{P_A}$ and $\mathcal{P_D}$ together with the two sets: the history of execution $\mathcal{H}$ and the client's active credentials $\mathcal{C_P}$.

Our interactive access control solution for stateful services and applications is shown in Figure 8. The logical explanation of the algorithm is the following. The algorithm's input consists of client's sets of currently presented credentials $\mathcal{C}_p$, revoked ones $\mathcal{C}_r$ and the service request r. When

---

[5]excessing credentials can be also referred to as conflicting credentials

Global vars: $\mathcal{C}_\mathcal{N}, \mathcal{C}_\mathcal{M}, \mathcal{C}_\mathcal{E}$; Initially $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{E} = \emptyset$;
Input: $\mathcal{C}_p, \mathcal{C}_r$ and r; **Internal input**: $\mathcal{P}_\mathcal{A}, \mathcal{P}_\mathcal{D}, \mathcal{H}, \mathcal{C}_\mathcal{P}$;
Output: grant/deny/$<$ask($\mathcal{C}_\mathcal{M}$),revoke($\mathcal{C}_\mathcal{E}$)$>$;

1. update $\mathcal{C}_\mathcal{P} = (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_r) \cup \mathcal{C}_p$,
2. update $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{N} \cup (\mathcal{C}_\mathcal{M} \setminus \mathcal{C}_p)$, where $\mathcal{C}_\mathcal{M}$ is from the last interaction,
3. set up $\mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{E} = \emptyset$,
4. verify whether the request r is a security consequence of the policy access $\mathcal{P}_\mathcal{A}$ and the active credentials $\mathcal{C}_\mathcal{P}$, namely $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \not\models \bot$,
5. <u>if</u> the check succeeds <u>then</u> return **_grant_** <u>else</u>

   (a) compute the set of *disclosable credentials* $\mathcal{C}_\mathcal{D} = \{c \mid \mathcal{P}_\mathcal{D} \cup \mathcal{C}_\mathcal{P} \models c\} \setminus (\mathcal{C}_\mathcal{N} \cup \mathcal{C}_\mathcal{P})$,

   (b) use abduction to find a minimal set of *missing credentials* $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$ such that both $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M} \not\models \bot$,

   (c) <u>if</u> a set $\mathcal{C}_\mathcal{M}$ exists <u>then</u> return $<$**_ask($\mathcal{C}_\mathcal{M}$)_**,**_revoke($\mathcal{C}_\mathcal{E}$)_**$>$ and iterate <u>else</u>

      i. for every $c \in \mathcal{C}_\mathcal{P}$ introduce a new credential $\hat{c}$ in the language,

      ii. use abduction to find a minimal set of *missing credentials* $\mathcal{C}_\mathcal{M} \subseteq \{\hat{c} \mid c \in \mathcal{C}_\mathcal{P}\} \cup \mathcal{C}_\mathcal{D}$ such that
      - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \{c \leftarrow \text{not } \hat{c}. \mid c \in \mathcal{C}_\mathcal{P}\} \cup \mathcal{C}_\mathcal{M} \models r$,
      - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \{c \leftarrow \text{not } \hat{c}. \mid c \in \mathcal{C}_\mathcal{P}\} \cup \mathcal{C}_\mathcal{M} \not\models \bot$,

      iii. <u>if</u> no set $\mathcal{C}_\mathcal{M}$ exists <u>then</u> return **_deny_** <u>else</u>
         A. compute $\mathcal{C}_\mathcal{E} = \{c \mid \hat{c} \in \mathcal{C}_\mathcal{M}\}$ and $\mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{M} \cap \mathcal{C}_\mathcal{D}$,
         B. return $<$**_ask($\mathcal{C}_\mathcal{M}$)_**,**_revoke($\mathcal{C}_\mathcal{E}$)_**$>$ and iterate.

Figure 8: Interactive Access Control Algorithm for Stateful Services

a client requests a specific service the authorization mechanism creates a new session and initializes to empty sets the variables $\mathcal{C}_\mathcal{N}, \mathcal{C}_\mathcal{M}$ and $\mathcal{C}_\mathcal{E}$.

Then, the set of active credentials $\mathcal{C}_\mathcal{P}$ is updated by removing the revoked ones $\mathcal{C}_r$ from it and then adding the new credentials (ref. step 1). The declined credentials $\mathcal{C}_\mathcal{N}$ are updated by credentials the client was asked in the previous interaction minus the ones that he has currently presented. Step 3 prepares the two sets $\mathcal{C}_\mathcal{M}$ and $\mathcal{C}_\mathcal{E}$ for the interaction output.

After the client's profile is updated, the algorithm checks whether the request r is granted by $\mathcal{P}_\mathcal{A}$ according to the client's set of active credentials $\mathcal{C}_\mathcal{P}$ (step 4). If the check fails then in step 5a the algorithm computes all credentials disclosable from $\mathcal{P}_\mathcal{D}$ and $\mathcal{C}_\mathcal{P}$ and from the resulting set removes all already declined and presented credentials. In this way we avoid dead loops of asking something already declined or presented. Step 5b computes (using abduction reasoning) a (minimal) solution for r. This is essentially our own interactive access control algorithm for stateless services presented in §8.

If in step 7c no $\mathcal{C}_\mathcal{M}$ exists then we come to the part of the algorithm devoted to stateful systems. The motivation here is that if a solution for r cannot be found in $\mathcal{C}_\mathcal{D}$ it means that

- either the client *does not have enough privileges* to get the disclosure of more missing credentials so that the abduction can find a solution

- or in the client's set of credentials $\mathcal{C}_\mathcal{P}$ there is *something "wrong"* that bans the client to get any solution, i.e. it makes $\mathcal{P}_\mathcal{A}$ inconsistent.

In the first case there is nothing you can do and we should just quietly deny access. In the second case we could have a possibility for recovery.

So, following the second case, in steps 5(c)i and 5(c)ii, we use abduction over the set of disclosable and active credentials $\mathcal{C}_\mathcal{D} \cup \mathcal{C}_\mathcal{P}$ searching for a possible solution $\mathcal{C}_\mathcal{M}$ that unlocks r and preserves consistency in $\mathcal{P}_\mathcal{A} \cup \mathcal{H}$. If a solution for r is found it clearly indicates that this solution could not be found in step 5b because of the existence of "wrong" credentials in $\mathcal{C}_\mathcal{P}$ that makes $\mathcal{P}_\mathcal{A}$ inconsistent. In this case we compute the set of excessing credentials $\mathcal{C}_\mathcal{E}$ as the set difference $\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{M}$. Here we separate the definitely good (consistent) solution $\mathcal{C}_\mathcal{M}$ from the rest in $\mathcal{C}_\mathcal{P}$.

Notice that steps 5(c)i–5(c)iii could be simplified by simply setting $\mathcal{C}_\mathcal{E} = \mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{P} = \emptyset$, i.e. asking the client to revoke everything and restart from scratch. We believe that this is hardly practical. We want to have a more precise control on the revokable credentials i.e. of being able to compute a minimal set of revokable and missing credentials. To do so we introduce for each credential $c \in \mathcal{C}_\mathcal{P}$ a new symbol for it $\hat{c}$ in the model, step 5(c)i in Figure 8. Then after obtaining the set of new symbols $\{\hat{c} \mid c \in \mathcal{C}_\mathcal{P}\}$ we generate a set of rules $\{c \leftarrow \text{not } \hat{c}. \mid c \in \mathcal{C}_\mathcal{P}\}$. The trick here is that negating all credentials in $\mathcal{C}_\mathcal{P}$ using the newly introduced symbols and running abduction reasoning over the set union of $\{\hat{c} \mid c \in \mathcal{C}_\mathcal{P}\} \cup \mathcal{C}_\mathcal{D}$ (step 5(c)ii) allows us to find a *minimal* solution $\mathcal{C}_\mathcal{M}$ for r that itself indicates what should be revoked and what should be asked from the client.

Let us consider the set $\{c \leftarrow \text{not } \hat{c}. \mid c \in \mathcal{C}_\mathcal{P}\}$. Since we attach this set to $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P}$ so it follows that all credentials in $\mathcal{C}_\mathcal{P}$ will be deduced again except those that the corresponding new symbol appears in $\mathcal{C}_\mathcal{M}$. So, all new-symbol credentials appearing in $\mathcal{C}_\mathcal{M}$, computed in step 7(c)ii, will be treated such that the absence of their respective credentials in $\mathcal{C}_\mathcal{P}$ allows the abduction reasoning to find a solution set for r. The solution itself may contain credentials from $\mathcal{C}_\mathcal{D}$ and if so these credentials should be asked as missing ones from the client.

Remark 4 (Multiple Activations and Revocations of Credentials) *In an interactive access control process the algorithm may ask the client to present credentials that he has revoked (was explicitly asked for that) in previous interactions or ask the client to revoke credentials that the same has already activated.*

Since we do not know what solution a client has for a particular resource so in the presence of alternatives in the access policy the system may choose the "wrong" one[6] such that later on when the right alternative is chosen it may require the revocation/activation of credentials that were already activated/revoked by the client in the interactions with the "wrong" one.

Example 5 *Abstracting from a specific meaning and for the sake of simplicity let us have the following scenario. A client with a set of available credentials $\{C_A, C_B, C_C\}$ wants to access a service r. The client's set of active credentials (already presented to the system) is $\mathcal{C}_\mathcal{P} = \{C_C\}$ and history $\mathcal{H} = \emptyset$. The policies for access and disclosure control are shown below.*

Access Policy $\mathcal{P}_\mathcal{A}$	Disclosure Policy $\mathcal{P}_\mathcal{D}$
$r \leftarrow C_A, C_B.$	$C_A \leftarrow .$
$r \leftarrow C_C, C_D.$	$C_B \leftarrow .$
$\leftarrow C_A, C_C.$	$C_C \leftarrow .$
	$C_D \leftarrow .$

Now let suppose that the client initially requests service r with set of presented credentials $\mathcal{C}_p = \{C_A\}$.

---

[6]Here we call "wrong" alternative a solution set that the client does not have it.

Then according to the algorithm in Figure 8 the check in step 4 will fail, then in step 5b abduction will not find any solution because the policy is inconsistent with the client's set of credentials and so the algorithm will reach step 5(c)ii. The variables and their respective values are listed in the table below.

$\mathcal{H}$	$\mathcal{C_P}$	$\mathcal{P_A} \cup \mathcal{H} \cup \{c \leftarrow not\ \hat{c}.\ \mid\ c \in \mathcal{C_P}\}$	$\mathcal{C_D}$	$\mathcal{C_N}$	$\{\hat{c}\ \mid\ c \in \mathcal{C_P}\} \cup \mathcal{C_D}$
$\emptyset$	$\{C_A, C_C\}$	$r \leftarrow C_A, C_B.$ $r \leftarrow C_C, C_D.$ $\leftarrow C_A, C_C.$ $C_A \leftarrow not\ \hat{C}_A.$ $C_C \leftarrow not\ \hat{C}_C.$	$\{C_B, C_D\}$	$\emptyset$	$\{\hat{C}_A, \hat{C}_C, C_B, C_D\}$

The output of this step considering the minimality criterion subset containment is:

- *Abduction output: $\{\hat{C}_A, C_D\}$, algorithm result: $\mathbf{ask}(\{C_D\})$, $\mathbf{revoke}(\{C_A\})$*

- *Abduction output: $\{\hat{C}_C, C_B\}$, algorithm result: $\mathbf{ask}(\{C_B\})$, $\mathbf{revoke}(\{C_C\})$*

Here it comes the point where Remark 4 takes place. Since we do not know what credentials and especially what solution the client has in possession we must choose one of the two outcomes listed above.

If we are lucky and choose the solution $<\mathbf{ask}(\{C_B\}), \mathbf{revoke}(\{C_C\})>$ then on the next interaction since the client has in possession the set $\{C_A, C_B, C_C\}$ the same presents C_B and revokes C_C obtaining $\mathbf{grant}\ r$ in step 5 at the next interaction.

In the other case, if we choose the solution $<\mathbf{ask}(\{C_D\}), \mathbf{revoke}(\{C_A\})>$ then on the next interaction the client will revoke C_A but will decline to present C_D, simply because he does not have it. Then the check in step 4 will not succeed because $\mathcal{C_P} = \{C_C\}$ does not contain enough credentials to unlock r. Following that, the abduction reasoning is step 5b will not find a solution because in $\mathcal{C_P}$ there is a credentials C_C that is inconsistent with the only solution available in $\mathcal{C_D} = \{C_A, C_B\}$.

Running again abduction reasoning with minimality according subset containment the only solution found is $\{\hat{C}_C, C_A, C_B\}$ and the respective outcome of the algorithm is $\mathbf{ask}(\{C_A, C_B\})$, $\mathbf{revoke}(\{C_C\})$. Essentially, we ask the client to restart from scratch, i.e. to revoke all of his active credentials ($\mathcal{C_E} = \mathcal{C_P} = \{C_C\}$) such that he can start from an initial (consistent) state of the system.

On the next interaction since the client has in possession $\{C_A, C_B, C_C\}$ so he revokes C_C, presents $\{C_A, C_B\}$ and in step 5 gets $\mathbf{grant}\ r$. □

9.3 Coping with Malicious Clients

We need to improve the algorithm in Figure 8 to protect the server against Denial of Service (DoS) attacks. To do so we consider the client as an entity that can manipulate the system only via its input sets of credentials $\mathcal{C}_p$ and $\mathcal{C}_r$. Particularly, he may present in his input set $\mathcal{C}_p$ credentials, which he has revoked in past interactions with the system, without been explicitly asked for it and, respectively, may revoke credentials in $\mathcal{C}_r$, which he has activated and presented to the system in past interactions, again without been asked for it. In both cases it would turn the system in a previous state (by letting it compute the same solution again and again). A malicious client could thus waste server's time forever.

Example 6 (Malicious Behavior) *Considering Example 5 with the same initial conditions let us play the following scenario. Again the client submits C_A when initially requests r. Then on*

Global vars: $\mathcal{C}_\mathcal{N}, \mathcal{C}_\mathcal{R}, \mathcal{C}_\mathcal{U}, \mathcal{C}_\mathcal{M}, \mathcal{C}_\mathcal{E}$;   Initially $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{R} = \mathcal{C}_\mathcal{U} = \mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{E} = \emptyset$;
Input: $\mathcal{C}_p$, $\mathcal{C}_r$ and r;   **Internal input**: $\mathcal{P}_\mathcal{A}, \mathcal{P}_\mathcal{D}, \mathcal{H}, \mathcal{C}_\mathcal{P}$;
Output: grant/deny/$<$ask($\mathcal{C}_\mathcal{M}$),revoke($\mathcal{C}_\mathcal{E}$)$>$;

1. update $\mathcal{C}_\mathcal{R} = (\mathcal{C}_\mathcal{R} \setminus \mathcal{C}_\mathcal{M}) \cup (\mathcal{C}_r \cap \mathcal{C}_\mathcal{E})$,
2. update $\mathcal{C}_\mathcal{P} = (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{R}) \cup (\mathcal{C}_p \setminus \mathcal{C}_\mathcal{R}) \cup (\mathcal{C}_p \cap \mathcal{C}_\mathcal{M}) \cup (\mathcal{C}_p \cap \mathcal{C}_\mathcal{N})$,
3. update $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{N} \cup (\mathcal{C}_\mathcal{M} \setminus \mathcal{C}_p)$,
4. update $\mathcal{C}_\mathcal{U} = \mathcal{C}_\mathcal{U} \cup (\mathcal{C}_\mathcal{E} \setminus \mathcal{C}_r)$,
5. set up $\mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{E} = \emptyset$,
6. verify whether $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \not\models \bot$,
7. <u>if</u> the check succeeds <u>then</u> return **grant** <u>else</u>

   (a) compute the set of *disclosable credentials*
   $\mathcal{C}_\mathcal{D} = \{c \mid \mathcal{P}_\mathcal{D} \cup \mathcal{C}_\mathcal{P} \models c\} \setminus (\mathcal{C}_\mathcal{N} \cup \mathcal{C}_\mathcal{P})$,
   (b) use abduction to find a minimal set of *missing credentials* $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$ such that both $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M} \not\models \bot$,
   (c) <u>if</u> a set $\mathcal{C}_\mathcal{M}$ exists <u>then</u> return $<$**ask($\mathcal{C}_\mathcal{M}$),revoke($\mathcal{C}_\mathcal{E}$)**$>$ and iterate <u>else</u>

      i. for every $c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U})$ introduce a new credential $\hat{c}$ in the language,
      ii. use abduction to find a minimal set of *missing credentials* $\mathcal{C}_\mathcal{M} \subseteq \{\hat{c} \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U})\} \cup \mathcal{C}_\mathcal{D}$ such that

         - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup (\mathcal{C}_\mathcal{P} \cap \mathcal{C}_\mathcal{U}) \cup \{c \leftarrow \text{not } \hat{c}. \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U})\} \cup \mathcal{C}_\mathcal{M} \models r$,
         - $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup (\mathcal{C}_\mathcal{P} \cap \mathcal{C}_\mathcal{U}) \cup \{c \leftarrow \text{not } \hat{c}. \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U})\} \cup \mathcal{C}_\mathcal{M} \not\models \bot$,
      iii. <u>if</u> no set $\mathcal{C}_\mathcal{M}$ exists <u>then</u> return **deny** <u>else</u>
         A. compute $\mathcal{C}_\mathcal{E} = \{c \mid \hat{c} \in \mathcal{C}_\mathcal{M}\}$ and $\mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{M} \cap \mathcal{C}_\mathcal{D}$,
         B. return $<$**ask($\mathcal{C}_\mathcal{M}$),revoke($\mathcal{C}_\mathcal{E}$)**$>$ and iterate.

Figure 9: Extended Interactive Access Control Algorithm

the next step the algorithm in Figure 8 returns $<ask(C_B)$, $revoke(C_C)>$ which is one of the two alternatives at this step and we refer to Example 5 for details. Next, since the client is a malicious one, he decides to present C_B but declines to revoke C_C in the next interaction. So, the client's set of active credentials becomes $\mathcal{C}_\mathcal{P} = \{C_A, C_B, C_C\}$ and the algorithm's output in this interaction step is $<ask(\emptyset)$, $revoke(C_C)>$.

Observing the algorithm's behavior, now the client decides to revoke not only credential C_C but also his initially submitted credential C_A. After the update in the next step, the client's set of active credentials becomes $\mathcal{C}_\mathcal{P} = \{C_B\}$. According to minimality criterion subset containment the algorithm returns $<ask(C_A)$, $revoke(\emptyset)>$ which immediately hints the client that the set of three credentials C_A, C_B, C_C makes the system state inconsistent. So, at this point the client can easily turn the system in a previous state by presenting C_A and C_C and the expected response by the algorithm on the next interaction is $<ask(\emptyset)$, $revoke(C_C)>$. □

The new algorithm is shown in Figure 9. In step 1, a new data set is introduced, the set of *revoked credentials* $\mathcal{C}_\mathcal{R}$, which accumulates all credentials revoked by a client in an interaction session for a particular service r. So, step 1 updates the set of revoked credentials by removing from $\mathcal{C}_\mathcal{R}$ the set of missing credentials $\mathcal{C}_\mathcal{M}$, asked in the last interaction, and adding to the resulting set the set of newly revoked credentials $\mathcal{C}_r$. The motivation for the set difference $\mathcal{C}_\mathcal{R} \setminus \mathcal{C}_\mathcal{M}$ is that

whatever the client presents from $\mathcal{C}_\mathcal{M}$ it is dropped from $\mathcal{C}_\mathcal{R}$ because it is not any more revoked and whatever it is not presented in $\mathcal{C}_\mathcal{M}$ but is dropped from $\mathcal{C}_\mathcal{R}$ is added to the set of declined credentials $\mathcal{C}_\mathcal{N}$. In this case revoked and declined credentials are kept disjoint, i.e. $\mathcal{C}_\mathcal{R} \cap \mathcal{C}_\mathcal{N} = \emptyset$.

Extending further step 1, to prevent situations of revoking credentials not supposed to be revoked by the client, we update $\mathcal{C}_\mathcal{R}$ by adding only those credentials from $\mathcal{C}_r$ that the client was explicitly asked in the previous interaction, i.e. adding only $\mathcal{C}_r \cap \mathcal{C}_\mathcal{E}$.

Step 2 updates the client set of active credentials by removing from $\mathcal{C}_\mathcal{P}$ the set of all revoked credentials and adding to it the set of newly activated credentials $\mathcal{C}_p$. First we use the set difference $\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{R}$ to remove all revoked credentials and second, expanding the set of active credentials, we add from currently presented credentials $\mathcal{C}_p$ only the credentials that have not been revoked before $- \mathcal{C}_p \setminus \mathcal{C}_\mathcal{R} -$ and we add also those credentials in $\mathcal{C}_p$ that the system has asked the client $- \mathcal{C}_p \cap \mathcal{C}_\mathcal{M} -$ or the client has declined to present in past interactions but it is presenting now $- \mathcal{C}_p \cap \mathcal{C}_\mathcal{N}$.

In other words, step 2 allows the client to activate credentials among those:

- that the system has asked him to present in the last interaction or

- that he has denied to present in past interactions or

- brand new credentials[7] that the client has not supplied to the system at the time of interacting.

Then, in step 4, we introduce a new data set $\mathcal{C}_\mathcal{U}$. The role of $\mathcal{C}_\mathcal{U}$ is analogous of that for $\mathcal{C}_\mathcal{N}$ and serves as a data store for those credentials that a client has declined to revoke in an interaction session. $\mathcal{C}_\mathcal{U}$ is updated by adding to it the set difference of excessing credentials the client was asked in the last interaction minus the ones currently revoked. Similarly, once a client refuses to revoke a credential the same credential will not be considered in a possible output again.

We note that the client at any time can present credentials that he has declined to present in previous interactions, although he will never be asked for them again, but he is not allowed (the system will not consider) to revoke credentials that he has refused to revoke before without been asked for it. The last requirement is mainly because the revocation of credentials is usually a cumbersome process and once the client refuses to revoke a credential then it is unlikely to expect him to do it later in a negotiation process.

Another difference of the algorithm wrt a malicious behavior is in step 7(c)ii. Here we run the abduction reasoning over the set $\{\hat{c} \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U})\} \cup \mathcal{C}_\mathcal{D}$. The set difference $\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U}$ comes from the fact that we do not want to ask the client to revoke credentials that he has already refused to revoke. In this way we rule out those models where the client already denied to comply to. We also note that the two conditions in step 7(c)ii are analogous with their respective ones in Figure 8 because whatever we drop from $\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U}$ we add it by the intersection of $\mathcal{C}_\mathcal{P} \cap \mathcal{C}_\mathcal{U}$.

Now on, wherever we refer to the access control algorithm we refer to its extended model shown in Figure 9.

9.4 Completeness and Correctness

In the following we assume that the sets of missing and excessing credentials are disjoint, i.e. $\mathcal{C}_\mathcal{M} \cap \mathcal{C}_\mathcal{E} = \emptyset$, otherwise the client will reject the answer. Note that this is true for the interactive algorithm in Figure 9. We also assume that at any time in an interaction process the sets of currently presented and revoked credentials are disjoint, i.e. $\mathcal{C}_p \cap \mathcal{C}_r = \emptyset$.

---

[7]excluding the revoked credentials since the system is aware of them.

Definition 9.1 (Powerful and Compliant Client) *A powerful and compliant client is a client that whenever receives $<ask(\mathcal{C}_\mathcal{M}),revoke(\mathcal{C}_\mathcal{E})>$ returns $<\mathcal{C}_\mathcal{M},\mathcal{C}_\mathcal{E}>$, i.e. activates all $c \in \mathcal{C}_\mathcal{M}$ and revokes any $c \in \mathcal{C}_\mathcal{E}$.*

Definition 9.2 (Cooperative and Compliant Client) *A client with ability to manage (obtain or revoke) a set of credentials $\mathcal{C}$ is a cooperative and compliant client if whenever receives $<ask(\mathcal{C}_\mathcal{M}),revoke(\mathcal{C}_\mathcal{E}$ returns $<\mathcal{C}_\mathcal{M} \cap \mathcal{C}, \mathcal{C}_\mathcal{E} \cap \mathcal{C}>$, i.e. activates all $c \in (\mathcal{C}_\mathcal{M} \cap \mathcal{C})$ and revokes any $c \in (\mathcal{C}_\mathcal{E} \cap \mathcal{C})$.*

Proposition 9.1 *If a client has a set of credentials $\mathcal{C}$ then in all executions of the algorithm his set of active credentials $\mathcal{C}_\mathcal{P}$ never exceeds $\mathcal{C}$, i.e. $\mathcal{C}_\mathcal{P} \subseteq \mathcal{C}$.*

Proposition 9.2 *If a client has a set of credentials $\mathcal{C}$ then in all executions the excessing credentials $\mathcal{C}_\mathcal{E}$, he could be potentially asked, are among those in $\mathcal{C}$, i.e. $\mathcal{C}_\mathcal{E} \subseteq \mathcal{C}$.*

Proof. The proof is trivial and it follows straightforwardly from the algorithm's structure. Step 7(c)iiiA computes excessing credentials from the set of active credentials $\mathcal{C}_\mathcal{P}$, i.e. $\mathcal{C}_\mathcal{E} \subseteq \mathcal{C}_\mathcal{P}$. Then according to Proposition 9.1 follows that $\mathcal{C}_\mathcal{E} \subseteq \mathcal{C}$. □

Proposition 9.3 *For any client with the set of credentials $\mathcal{C}$ is valid that $\mathcal{C}_\mathcal{E} \cap \mathcal{C} = \mathcal{C}_\mathcal{E}$.*

Lemma 9.1 *Let $\mathcal{P}_\mathcal{A}$ be an access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy and r a request. If the algorithm in Figure 9 returns $<ask(\mathcal{C}_\mathcal{M}), revoke(\mathcal{C}_\mathcal{E})>$ then either $\mathcal{C}_\mathcal{M} \neq \emptyset$ or $\mathcal{C}_\mathcal{E} \neq \emptyset$.*

Proof. We will prove the claim in two parts. First part considering the output $<ask(\mathcal{C}_\mathcal{M}),$ revoke$(\mathcal{C}_\mathcal{E})>$ returned in step 7c and showing that it is different than $<ask(\emptyset), revoke(\emptyset)>$ and the second part showing the same but for the output in step 7(c)iiiB.

Proving the first part is rather straightforward and it follows from the fact that if we assume that $\mathcal{C}_\mathcal{M} = \emptyset$ (no need of extra access rights) then in the step 7b before we contradict with the fact that the client does not have enough access rights. So, $\mathcal{C}_\mathcal{M} \neq \emptyset$.

The second part we will prove by again assuming the converse and showing that it contradicts with some facts. So, let assume that the algorithm in step 7(c)ii finds $\mathcal{C}_\mathcal{M}$ and in step 7(c)iiiA computes $\mathcal{C}_\mathcal{E} = \emptyset$. It means that there is *no* new-symbol credential $\hat{c} \in \mathcal{C}_\mathcal{M}$, where $\mathcal{C}_\mathcal{M}$ is computed in step 7(c)ii. Therefore $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$ thus we have that $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup (\mathcal{C}_\mathcal{P} \cap \mathcal{C}_\mathcal{U}) \cup \{c \leftarrow \text{not } \hat{c} \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{U})\} \cup \mathcal{C}_\mathcal{M} \models r$. However, since $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$ this implies that $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup (\mathcal{C}_\mathcal{P} \cap \mathcal{C}_\mathcal{U}) \cup \{c \leftarrow \text{not } \hat{c} \mid c \in (\mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{U})\} \cup \mathcal{C}_\mathcal{M} \models c$ for all $c \in \mathcal{C}_\mathcal{P}$ because $\hat{c}$ is a new symbol not occurring in $\mathcal{P}_\mathcal{A}$ and $\mathcal{H}$. Therefore, $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M} \not\models \bot$ which contradicts with the assumption in step 7b that no such set $\mathcal{C}_\mathcal{M}$ exists. □

Theorem 9.1 (Soundness) *Let $\mathcal{P}_\mathcal{A}$ be an access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy and r a request. If a client gets grant r with the algorithm in Figure 9 then he has a solution set $\mathcal{C}_\mathcal{S}$ that unlocks r according to $\mathcal{P}_\mathcal{A}$.*

Proof. This proof is rather straightforward. Let suppose that the client, requested service r, got *grant*. The only way to get it is when $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{P} \models r$. Then there are two cases: either $\mathcal{C}_\mathcal{P} = \emptyset$ or $\mathcal{C}_\mathcal{P} \neq \emptyset$.

If $\mathcal{C}_\mathcal{P} = \emptyset$ then the resource r is not protected by $\mathcal{P}_\mathcal{A}$ or must be executed by conditions related to $\mathcal{H}$. In both cases $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \models r$. So, the $\emptyset$ is a solution for r and the client has it.

If $\mathcal{C}_\mathcal{P} \neq \emptyset$ then the only way to introduce a credential in $\mathcal{C}_\mathcal{P}$ is by step 2 of the algorithm. Since initially $\mathcal{C}_\mathcal{P} = \emptyset$ so the client has sent a sequence of sets of credentials $\mathcal{C}_{p_1}, \ldots, \mathcal{C}_{p_n}$ such that $\bigcup_{i=1}^{n} \mathcal{C}_{p_i} = \mathcal{C}_\mathcal{P}$. Then the client has a set of credentials that unlocks it. □

Theorem 9.2 (Termination) *Let $\mathcal{P}_\mathcal{A}$ be an access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy and r a request. The access control algorithm in Figure 9 always terminates.*

Proof. The algorithm terminates when either *grant* or *deny* is returned back.

We will prove the termination by well-founded tuple ordering. At each interaction we associate a tuple $<\mathcal{C}_1,\ldots,\mathcal{C}_n>$ of credentials and stipulate that $<\mathcal{C}_1,\ldots,\mathcal{C}_n> \prec <\mathcal{C}_1^{'},\ldots,\mathcal{C}_n^{'}>$ if either $\mathcal{C}_1 \subset \mathcal{C}_1^{'}$ or for some $i < n$ $\mathcal{C}_1 = \mathcal{C}_1^{'}$, $\mathcal{C}_2 = \mathcal{C}_2^{'}$, $\ldots$, $\mathcal{C}_i = \mathcal{C}_i^{'}$ and $\mathcal{C}_{i+1} \subset \mathcal{C}_{i+1}^{'}$.

This is a well-founded ordering which has as a top element the tuple with all credentials occurring in the policies. We associate the tuple $<\mathcal{C}_\mathcal{N}, \mathcal{C}_\mathcal{U}, \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R}>$ to each interaction. At the end of the interaction, which does not return grant or deny, we have already proved by Lemma 9.1 that $\mathcal{C}_\mathcal{M} \neq \emptyset$ or $\mathcal{C}_\mathcal{E} \neq \emptyset$. At the next interaction we have also received $\mathcal{C}_p$ and $\mathcal{C}_r$. We use $<\mathcal{C}_\mathcal{N}, \mathcal{C}_\mathcal{U}, \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R}>$ to denote the previous interaction and the primed version for the values at the end of the current interaction.

If $\mathcal{C}_\mathcal{M} \setminus \mathcal{C}_p \neq \emptyset$ then $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{N} \cup (\mathcal{C}_\mathcal{M} \setminus \mathcal{C}_p)$ is strictly increasing and thus $<\mathcal{C}_\mathcal{N}, \mathcal{C}_\mathcal{U}, \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R}> \prec <\mathcal{C}_\mathcal{N} \cup (\mathcal{C}_\mathcal{M} \setminus \mathcal{C}_p), \mathcal{C}_\mathcal{U}^{'}, \mathcal{C}_\mathcal{P}^{'} \cup \mathcal{C}_\mathcal{R}^{'}>$ and we are done.

if $\mathcal{C}_\mathcal{M} \setminus \mathcal{C}_p = \emptyset$ and $\mathcal{C}_\mathcal{E} \setminus \mathcal{C}_r \neq \emptyset$ then $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{N} \cup (\mathcal{C}_\mathcal{M} \setminus \mathcal{C}_p)$ is unchanged but $\mathcal{C}_\mathcal{U} = \mathcal{C}_\mathcal{U} \cup (\mathcal{C}_\mathcal{E} \setminus \mathcal{C}_r)$ is strictly increasing and thus $<\mathcal{C}_\mathcal{N}, \mathcal{C}_\mathcal{U}, \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R}> \prec <\mathcal{C}_\mathcal{N} \cup (\mathcal{C}_\mathcal{M} \setminus \mathcal{C}_p), \mathcal{C}_\mathcal{U} \cup (\mathcal{C}_\mathcal{E} \setminus \mathcal{C}_r), \mathcal{C}_\mathcal{P}^{'} \cup \mathcal{C}_\mathcal{R}^{'}>$ and we are done.

Let us now consider the case $\mathcal{C}_\mathcal{M} \setminus \mathcal{C}_p = \emptyset$ and $\mathcal{C}_\mathcal{E} \setminus \mathcal{C}_r = \emptyset$. If $\mathcal{C}_r = \mathcal{C}_\mathcal{E}$ and $\mathcal{C}_\mathcal{M} = \mathcal{C}_p$ then by construction the algorithm returns grant. So, we must be in a situation where $\mathcal{C}_r \supset \mathcal{C}_\mathcal{E}$ or $\mathcal{C}_p \supset \mathcal{C}_\mathcal{M}$, i.e. we have either given something we have not been asked or revoked something we have not been asked to revoke.

Since $\mathcal{C}_\mathcal{N}$ and $\mathcal{C}_\mathcal{U}$ remain unchanged therefore we have to prove that $\mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R} \subset \mathcal{C}_\mathcal{P}^{'} \cup \mathcal{C}_\mathcal{R}^{'}$ where according to Figure 9 we have

$$(1) \quad \mathcal{C}_\mathcal{R}^{'} = (\mathcal{C}_\mathcal{R} \setminus \mathcal{C}_\mathcal{M}) \cup (\mathcal{C}_r \cap \mathcal{C}_\mathcal{E})$$
$$(2) \quad \mathcal{C}_\mathcal{P}^{'} = (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{R}^{'}) \cup (\mathcal{C}_p \setminus \mathcal{C}_\mathcal{R}^{'}) \cup (\mathcal{C}_p \cap \mathcal{C}_\mathcal{M}) \cup (\mathcal{C}_p \cap \mathcal{C}_\mathcal{N})$$

To prove it first we will show that $\forall c \in \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R} : c \in \mathcal{C}_\mathcal{P}^{'} \cup \mathcal{C}_\mathcal{R}^{'}$ and second that $\exists c \in \mathcal{C}_\mathcal{P}^{'} \cup \mathcal{C}_\mathcal{R}^{'} : c \notin \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R}$.

To prove that $\forall c \in \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R} : c \in \mathcal{C}_\mathcal{P}^{'} \cup \mathcal{C}_\mathcal{R}^{'}$ let us assume the converse $\exists c \in \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R} : c \notin \mathcal{C}_\mathcal{P}^{'} \cup \mathcal{C}_\mathcal{R}^{'}$ and we will show that it contradicts with some facts. Let $c \in \mathcal{C}_\mathcal{P}$ then since $c \notin \mathcal{C}_\mathcal{P}^{'}$ follows that according to step (2) $c \in \mathcal{C}_\mathcal{R}^{'}$ but then $c \in \mathcal{C}_\mathcal{P}^{'} \cup \mathcal{C}_\mathcal{R}^{'}$ which is contradiction.

Let $c \notin \mathcal{C}_\mathcal{P}$ and $c \in \mathcal{C}_\mathcal{R}$. Since $c \notin \mathcal{C}_\mathcal{R}^{'}$ and by step (1) follows that $c \notin \mathcal{C}_\mathcal{R} \setminus \mathcal{C}_\mathcal{M}$ so $c \in \mathcal{C}_\mathcal{M}$. Since $\mathcal{C}_p \supset \mathcal{C}_\mathcal{M}$ so $c \in \mathcal{C}_p \cap \mathcal{C}_\mathcal{M}$ and then from step (2) follows that $c \in \mathcal{C}_\mathcal{P}^{'}$. Thus $c \in \mathcal{C}_\mathcal{P}^{'} \cup \mathcal{C}_\mathcal{R}^{'}$ which is again contradiction.

We have just proved that $\forall c \in \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R} : c \in \mathcal{C}_\mathcal{P}^{'} \cup \mathcal{C}_\mathcal{R}^{'}$.

Let us now resume the conditions we have. We are in a case $\mathcal{C}_p \supset \mathcal{C}_\mathcal{M}$ or $\mathcal{C}_r \supset \mathcal{C}_\mathcal{E}$ where either $\mathcal{C}_p \supseteq \mathcal{C}_\mathcal{M}$ or $\mathcal{C}_r \supseteq \mathcal{C}_\mathcal{E}$.

To prove the second part that $\exists c \in \mathcal{C}_\mathcal{P}^{'} \cup \mathcal{C}_\mathcal{R}^{'} : c \notin \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R}$ we will divide it in the following cases and prove each of them respectively:

1. if $\mathcal{C}_r = \mathcal{C}_\mathcal{E}$ and $\mathcal{C}_p \setminus \mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{R}$ then we know that $\forall c \in \mathcal{C}_p \setminus \mathcal{C}_\mathcal{M}$ and step (2) follows that $c \notin \mathcal{C}_\mathcal{P}^{'}$ and thus the algorithm returns grant.

   This case describes the situation where a client presents as additional credentials some of those already revoked by him and so the system will not consider them when taking an access decision (they do not change $\mathcal{C}_\mathcal{P}$).

2. if $\mathcal{C}_r = \mathcal{C}_\mathcal{E}$ and $(\mathcal{C}_p \setminus \mathcal{C}_\mathcal{M}) \setminus \mathcal{C}_\mathcal{R} \subseteq \mathcal{C}_\mathcal{P}$ then we still get grant as $\forall c \in (\mathcal{C}_p \setminus \mathcal{C}_\mathcal{M}) \setminus \mathcal{C}_\mathcal{R}$ holds that $c \notin \mathcal{C}_\mathcal{R}'$ (ref. step (1)) and following that we get $c \in \mathcal{C}_\mathcal{P}'$ (ref. step (2)), but it was still previously consistent with the policy and so the algorithm returns grant.

   Here we extend case 1 and reason for those credentials, additionally presented by the client, that have already been activated by him. In this case these credentials do not influence on the access decision since they do not change $\mathcal{C}_\mathcal{P}$.

3. if $\mathcal{C}_r = \mathcal{C}_\mathcal{E}$ and $\mathcal{C}_p \setminus \mathcal{C}_\mathcal{M} \not\subseteq \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R}$ then $\exists c \in \mathcal{C}_p \setminus \mathcal{C}_\mathcal{M} : c \notin \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R}$ then according to step (1) and assumption that $\mathcal{C}_p \cap \mathcal{C}_r = \emptyset$ follows that $c \notin \mathcal{C}_\mathcal{R}'$ and so in step (2) this credential is added to $\mathcal{C}_\mathcal{P}'$. Therefore it holds that $\mathcal{C}_\mathcal{P}' \cup \mathcal{C}_\mathcal{R}' \supset \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{R}$.

4. if $\mathcal{C}_r \supset \mathcal{C}_\mathcal{E}$ then $\forall c \in \mathcal{C}_r \setminus \mathcal{C}_\mathcal{E}$ follows that

   - if $c \notin \mathcal{C}_\mathcal{R} \setminus \mathcal{C}_\mathcal{M}$ then $c \notin \mathcal{C}_\mathcal{R}'$ and so it does not change $\mathcal{C}_\mathcal{P}'$ and thus the proof follows from the previous cases 1, 2 and 3 with $\mathcal{C}_r = \mathcal{C}_\mathcal{E}$.
   - if $c \in \mathcal{C}_\mathcal{R} \setminus \mathcal{C}_\mathcal{M}$ then it will also not change $\mathcal{C}_\mathcal{R}'$ and $\mathcal{C}_\mathcal{P}'$ from the previous interaction and we can go to the previous cases 1, 2 and 3 with $\mathcal{C}_r = \mathcal{C}_\mathcal{E}$.

So, we proved that in an interaction process a client either increases the set of declined credentials $\mathcal{C}_\mathcal{N}$ or, if it remains unchanged, he increases the set of not revoked credentials $\mathcal{C}_\mathcal{U}$ or, if both of them remain unchanged, the client increases his sets of total credentials[8] submitted to the system. The process continues until the client exhausts all credentials occurring in the access policy.

With this we finish the proof. $\square$

9.5 Completeness for Powerful and Cooperative Clients

Theorem 9.3 (Completeness for a Monotonic Access Policy) *Let $\mathcal{P}_\mathcal{A}$ be a monotonic access policy, $\mathcal{P}_\mathcal{D}$ be a monotonic disclosure policy and r a request. If $\mathcal{P}_\mathcal{A}$ and $\mathcal{P}_\mathcal{D}$ guarantee fair access and interaction then a powerful or cooperative client (as defined in §.8.1 of the basic framework) always gets* grant *r with the algorithm in Figure 9.*

Proof. Essentially along the lines of the algorithm in Figure 6. $\square$

Of course the whole business of stateful systems requires non-monotonic policies, so this result is not enough. Here we relax the policy access $\mathcal{P}_\mathcal{A}$ from the assumption of monotonic policy and consider it as an arbitrary non-monotonic policy. So, from now on we assume that $\mathcal{P}_\mathcal{A}$ is *non-monotonic* and $\mathcal{P}_\mathcal{D}$ *monotonic* unless explicitly specified otherwise.

In the same context, we inherit and extend the assumption from the basic framework so that, now, whenever a client initially requests a service he submits $\mathcal{C}_p = \mathcal{C}_r = \emptyset$ and if hidden credentials $\mathcal{C}_\mathcal{H}$ needed then $\mathcal{C}_p = \mathcal{C}_\mathcal{H}$ and $\mathcal{C}_r = \emptyset$.

Here one can doubt why we do not relieve the assumption of arbitrary sets of initially presented and revoked credentials. Anyway it will not influence on the proofs since whatever is in the initial sets $\mathcal{C}_p$ and $\mathcal{C}_r$ we can add $\mathcal{C}_p$ to $\mathcal{C}_\mathcal{P}$ and drop $\mathcal{C}_r$ from $\mathcal{C}_\mathcal{P}$ and restart the proof with the new set $\mathcal{C}_\mathcal{P}$ assuming that the client initially presents $\mathcal{C}_p = \mathcal{C}_r = \emptyset$.

---

[8]Credentials that are either revoked or active.

Theorem 9.4 (Completeness for a Powerful and Compliant Client) *Let $\mathcal{P}_\mathcal{A}$ be an access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy, $\mathcal{H}$ be history of executions and r a request. If $\mathcal{P}_\mathcal{A} \cup \mathcal{H}^9$ and $\mathcal{P}_\mathcal{D}$ guarantee fair access and interaction then a powerful and compliant client always gets grant r with the algorithm in Figure 9.*

    Proof. Let us have a powerful client that requests r with an initial set of presented and revoked credentials equal to an empty set, namely $\mathcal{C}_p = \mathcal{C}_r = \emptyset$. Also we assume that the set of active credentials is a non-empty set containing credentials that the client has submitted in previous interactions with services within the same partner's domain, $\mathcal{C}_\mathcal{P} \neq \emptyset$.

    So, keeping the assumption of the initial empty sets, steps 1 to 5 compute $\mathcal{C}_p = \mathcal{C}_r = \mathcal{C}_\mathcal{R} = \mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{U} = \mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{E} = \emptyset$. If the check in step 6 succeeds then in step 7 the algorithm returns grant (either no credentials are needed for r or the client already has a solution in $\mathcal{C}_\mathcal{P}$ for r) and we are done.

    If step 6 does not succeed then the algorithm goes to step 7a. Then using the assumption of fair access and interaction follows that a solution $\mathcal{C}_\mathcal{S}$ for r exists, it is disclosable by $\mathcal{P}_\mathcal{D}$ and $(\mathcal{C}_\mathcal{S} \setminus \mathcal{C}_\mathcal{P}) \subseteq \mathcal{C}_\mathcal{D}$.

    Now that we are sure of existence of a solution for r there are two cases:

- Step 7b finds a missing set $\mathcal{C}_\mathcal{M}$ and so $<\boldsymbol{ask(\mathcal{C}_\mathcal{M})}, \boldsymbol{revoke(\mathcal{C}_\mathcal{E})}>$ is sent back to the client. In this case $\mathcal{C}_\mathcal{E} = \emptyset$.

  Then on the next interaction step, since the client is a powerful one, he presents all $c \in \mathcal{C}_\mathcal{M}$. After the updates, in step 6, since $\mathcal{C}_\mathcal{P} = \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M}$ then the check succeeds and in the next step 7 the algorithm returns *grant r*.

- Step 7b does not find a missing set $\mathcal{C}_\mathcal{M}$. In this case, since at least one solution $\mathcal{C}_\mathcal{S}$ exists in $\mathcal{C}_\mathcal{D}$, follows that in $\mathcal{C}_\mathcal{P}$ there are "wrong" credentials that ban the client to get a solution, namely $\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{S} \neq \emptyset$. So, the algorithm goes to step 7(c)ii. At this step the abduction reasoning does compute a missing set $\mathcal{C}_\mathcal{M}$ simply because, at least, one such exists and is $\mathcal{C}_\mathcal{M} = \{\hat{c} \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{S})\} \cup (\mathcal{C}_\mathcal{S} \setminus \mathcal{C}_\mathcal{P})$ such that $\mathcal{C}_\mathcal{M} \subseteq \{\hat{c} \mid c \in \mathcal{C}_\mathcal{P}\} \cup \mathcal{C}_\mathcal{D}$ and $\mathcal{C}_\mathcal{E} = \{c \mid \hat{c} \in \mathcal{C}_\mathcal{M}\} \neq \emptyset$ conforming to Lemma 9.1. We have the set difference of $\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{S}$ and $\mathcal{C}_\mathcal{S} \setminus \mathcal{C}_\mathcal{P}$ just to assure that we do not ask the client to revoke and, at the same time, activate any credentials from the solution $\mathcal{C}_\mathcal{S}$.

  Here $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \{c \leftarrow \text{not } \hat{c}. \mid c \in \mathcal{C}_\mathcal{P}\} \cup \mathcal{C}_\mathcal{M}$ is equivalent to $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup \mathcal{C}_\mathcal{S}$. Since $\mathcal{C}_\mathcal{S}$ is a solution for r follows that abduction finds at least this missing set and step 7(c)iii is *skipped*. Then in step 7(c)iiiA the new $\mathcal{C}_\mathcal{E}$ and $\mathcal{C}_\mathcal{M}$ are computed. We remark that the just computed two sets replaced back in the equations of step 7(c)ii transform them into $\mathcal{P}_\mathcal{A} \cup \mathcal{H} \cup (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{E}) \cup \mathcal{C}_\mathcal{M} \models r$ and $\not\models \perp$.

  Then after the result $<\boldsymbol{ask(\mathcal{C}_\mathcal{M})}, \boldsymbol{revoke(\mathcal{C}_\mathcal{E})}>$ is returned back to the client, since he is powerful, he activates all missing and revokes all excessing credentials. On the next interaction, in step 6, since $\mathcal{C}_\mathcal{P}$ is updated to $\mathcal{C}_\mathcal{P} = (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{E}) \cup \mathcal{C}_\mathcal{M}$ follows that the deduction check succeeds and in step 7 the algorithm returns *grant r*.

We note that the algorithm grants a powerful client in no more than two interactions. With this we finish the proof. $\qquad\square$

---

    [9]It is important to pose the property of fair access over $\mathcal{P}_\mathcal{A} \cup \mathcal{H}$ because it guarantees fairness wrt other (possibly) environment constraints like limited number of executions on a service, limited number of users accessing a service, etc.

Theorem 9.5 (Completeness for a Cooperative and Compliant Client) *Let $\mathcal{P}_\mathcal{A}$ be an access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy, $\mathcal{H}$ be history of executions and r a request. If $\mathcal{P}_\mathcal{A} \cup \mathcal{H}$ and $\mathcal{P}_\mathcal{D}$ guarantee fair access and interaction then if a cooperative and compliant client has a set of credentials $\mathcal{C}_\mathcal{S}$ that unlocks r according to $\mathcal{P}_\mathcal{A}$ then the client always gets grant r with the algorithm in Figure 9.*

Proof. We will prove the theorem in two parts. First part by induction, showing that in a single interaction if a cooperative client does not get *grant r* then he gets $<ask(\mathcal{C}_\mathcal{M}), revoke(\mathcal{C}_\mathcal{E})>$. In other words he will never receive denial of the algorithm. Second part, we will show that because of the first part and Theorem 9.2 for termination a cooperative client with a solution set $\mathcal{C}_\mathcal{S}$ always gets *grant r*.

Part I Proof by induction on the interaction steps:

Inter. 1: A cooperative client requests r with initial sets $\mathcal{C}_p = \mathcal{C}_r = \emptyset$. Since the client has a solution $\mathcal{C}_\mathcal{S}$ for r and because of fair access and interaction properties so at least one solution exists and is disclosable by $\mathcal{P}_\mathcal{D}$, i.e. $(\mathcal{C}_\mathcal{S} \setminus \mathcal{C}_\mathcal{P}) \subseteq \mathcal{C}_\mathcal{D}$.

Let assume that the client does not get *grant r* in step 7 of the algorithm and that abduction in step 7b cannot find a missing set $\mathcal{C}_\mathcal{M}$. Then the step 7(c)iii of denial is *skipped* because at least one solution exists, which is $\mathcal{C}_\mathcal{M} = \{\hat{c} \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{S})\} \cup (\mathcal{C}_\mathcal{S} \setminus \mathcal{C}_\mathcal{P})$. This solution satisfies the conditions in step 7(c)ii and can be checked in the same way as in the Theorem 9.4. After that steps 7(c)iiiA and 7(c)iiiB compute and return back to the client $<ask(\mathcal{C}_\mathcal{M}), revoke(\mathcal{C}_\mathcal{E})>$.

Inter. N: Here we use the abduction hypothesis that the client fails to get *grant r* and gets $<ask(\mathcal{C}_\mathcal{M}), revoke(\mathcal{C}_\mathcal{E})>$ in the previous interaction. Now, suppose that the client fails to get *grant r* in step 7 and in step 7b the abduction reasoning fails to find a missing set.

So, up to now in the client's set of $\mathcal{C}_\mathcal{P}$ there are "wrong" credentials that ban the client to get a missing set (solution) for r. Since the client has a solution $\mathcal{C}_\mathcal{S}$ for r and because of fair access and interaction properties so this solution is disclosable by $\mathcal{P}_\mathcal{D}$. Here is also the point where we use the fact that $\mathcal{P}_\mathcal{D}$ is a *monotonic* policy. It is because in the set of active credentials $\mathcal{C}_\mathcal{P}$ we have credentials partially compiled from requests for other solutions for r which the client does not have. And so if the policy $\mathcal{P}_\mathcal{D}$ is non-monotonic we cannot guarantee that the solution $\mathcal{C}_\mathcal{S}$ is always disclosable (in the presence of other credentials).

Again a missing set $\mathcal{C}_\mathcal{M} = \{\hat{c} \mid c \in (\mathcal{C}_\mathcal{P} \setminus \mathcal{C}_\mathcal{S})\} \cup (\mathcal{C}_\mathcal{S} \setminus \mathcal{C}_\mathcal{P})$ exists which analogously of Theorem 9.4 can be checked that it satisfies the conditions in step 7(c)ii and so denial is skipped. We only point out that since the client is cooperative then the sets $\mathcal{C}_\mathcal{U}$ and $\mathcal{C}_\mathcal{N}$ will never overlap with the credentials the client has presented to the system, i.e. $\mathcal{C}_\mathcal{U} \cap \mathcal{C}_\mathcal{P} = \emptyset$ and $\mathcal{C}_\mathcal{N} \cap \mathcal{C}_\mathcal{P} = \emptyset$. And so the result $<\mathbf{ask}(\mathcal{C}_\mathcal{M}), \mathbf{revoke}(\mathcal{C}_\mathcal{E})>$ in step 7(c)iiiB is returned back to the client.

Part II What we have so far:

- If the client does not get *grant* he gets $<\boldsymbol{ask(\mathcal{C}_\mathcal{M})}, \boldsymbol{revoke(\mathcal{C}_\mathcal{E})}>$ (just proved in Part I),

- The client is never idle, i.e. never gets $<\boldsymbol{ask(\emptyset)}, \boldsymbol{revoke(\emptyset)}>$ (follows from Lemma 9.1),

- The algorithm at certain interaction always returns *grant* or *deny* (Theorem 9.2),

According to the three cases above follows that the client in a finite number of interactions will get *grant r*. ☐

Theorem 9.6 (Completeness for a Powerful Client with Hidden Credentials) *Let $\mathcal{P}_\mathcal{A}$ be an access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy, $\mathcal{H}$ be history of executions and r a request. If $\mathcal{P}_\mathcal{A} \cup \mathcal{H}$ and $\mathcal{P}_\mathcal{D}$ guarantee fair access and interaction then a powerful client with hidden credentials $\mathcal{C}_\mathcal{H}$ for r always gets grant r with the algorithm in Figure 9.*

Proof. The proof of the theorem is analogously of that for Theorem 9.4. The only difference is that initially the client presents the set of hidden credentials $\mathcal{C}_\mathcal{H}$ that helps him to disclose the visible part of a solution $\mathcal{C}_\mathcal{S}$ from $\mathcal{P}_\mathcal{D}$, i.e. $\forall c \in \mathcal{C}_\mathcal{S} \setminus \mathcal{C}_\mathcal{H} : \mathcal{P}_\mathcal{D} \cup \mathcal{H} \models c$ (ref. Definitions 8.3 and 8.5). And so in step 7b because of fair access and interaction properties follows that $\mathcal{C}_\mathcal{S} \subseteq (\mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{D})$ and the abduction reasoning potentially can find a missing set $\mathcal{C}_\mathcal{M}$.

The rest of the proof follows (can be done along the same line) from Theorem 9.4. With this we finish the proof. ☐

Theorem 9.7 (Completeness for a Cooperative and Compliant Client with Hidden Credentials) *Let $\mathcal{P}_\mathcal{A}$ be an access policy, $\mathcal{P}_\mathcal{D}$ be a disclosure policy, $\mathcal{H}$ be history of executions and r a request. If $\mathcal{P}_\mathcal{A} \cup \mathcal{H}$ and $\mathcal{P}_\mathcal{D}$ guarantee fair access and interaction then if a cooperative and compliant client with hidden credentials $\mathcal{C}_\mathcal{H}$ for r has a solution set $\mathcal{C}_\mathcal{S}$ for $\mathcal{C}_\mathcal{H}$ then the client always gets grant r with the algorithm in Figure 9.*

Proof. Let us start from the fact that the client has a set of hidden credentials $\mathcal{C}_\mathcal{H}$ for r and the same has a solution set $\mathcal{C}_\mathcal{S}$ wrt $\mathcal{C}_\mathcal{H}$. It means that first according to Definition 8.3 the set $\mathcal{C}_\mathcal{H} \subseteq \mathcal{C}_\mathcal{S}$ – important step assuring that the client has the right set of hidden credentials *especially* for the solution $\mathcal{C}_\mathcal{S}$. Second, according to Definition 8.5 the solution set $\mathcal{C}_\mathcal{S}$ is disclosable by $\mathcal{P}_\mathcal{D}$ and $\mathcal{C}_\mathcal{H}$.

Then using the assumption on the client with hidden credentials, that whenever he requests a service he initially presents the hidden credentials, we can construct the same proof along the one for Theorem 9.5. The only difference is that since the hidden credentials are submitted initially so at least the solution $\mathcal{C}_\mathcal{S}$ is disclosable by $\mathcal{P}_\mathcal{D}$ and using fair access and interaction properties we always return $<ask(\mathcal{C}_\mathcal{M}), revoke(\mathcal{C}_\mathcal{E})>$ until the client discloses the entire $\mathcal{C}_\mathcal{S}$.

With this we finish the proof. ☐

10 Related Work

As we mentioned in the introduction, access control for autonomic communication borrows some aspect of trust management and some aspects of workflow security. Among these models we find a number of relevant works: for workflows [4], web services [32], role-based access control on the web [14, 33], tasks [16] and DRM [32], possibly coupled by sophisticated policy combination algorithms. However, they have mostly remained within the classical framework – all decisions of grant/deny are based on checking that request would follow from the policy and the presented set of credentials.

The work on trust negotiation [34] focuses on communication and infrastructure and assumes that requests and counter requests have been somehow calculated from the access policy. Also the formal models on credential-based access control and policy combination [4, 27, 15, 39] do not treat the problem of inferring missing credentials from failed requests. Also standardization effort like

XACML proposal [40] gives rules for deriving what is right (evaluating policies) and not rules for understanding what went wrong.

Also a recent proposal by Bonatti and Samarati [5] that has the explicit focus on access and release control is not fully on target. In a nutshell, the request is received, the policy rules are filtered for relevance, the relevant rules are partially evaluated and sent to the client. The client will have to figure out which are the credentials and then will evaluate these credentials according to its release policy.

The first problem is that demanding clients to analyze security policies is not acceptable here. The second problem is that after a suitable number of queries the entire policy of the server would be disclosed to the client. Here we only disclose the needed credentials and not the rules of the policy whose structure remains hidden to the client. Furthermore, the relevancy filtering approach only works for flat (monotone) policies, in which for every request we list all of its credentials.

The other key proposal on trust negotiation by Yu et al. [42], offers a dual view w.r.t Bonatti and Samarati [5]. Loosely speaking, each credential is associated to a policy (a boolean expression) denoting the credentials that a client must have already provided for its safe disclosure. By a step wise process the parties can exchange credentials or policy rules until the desired resource is released. The paper provides safe sequences of disclosure in a rather ad-hoc fashion building upon trees rather than logical formalization. As a consequence they can only treat monotone policies and it is not possible to define notions of consistency of policies and disclosure of policies in presence of constraints (e.g. separation of duty). Another limitation of the paper is that it interlocks the access and the release policy into one. So, as the authors acknowledge [42, page. 21], it is impossible to access resources if some of the needed credentials cannot be disclosed at some point. Furthermore, the need for intermediate credential disclosure calls for a structuring of policy rules that may be counter-intuitive from the point of view of access control. For instance, a policy rule may say that for access to the full text of an on-line journal article we must have already got access to browsing the journal's table of contents, plus additional credentials. Access to the table of contents could then specify some simpler set of credentials. For the disclosure process to take place such natural composition is not possible when using Yu et al. framework [42].

Another proposal by Yu and Winslett [41] postulates that policies for protecting resources should be themselves treated (protected) as first class sensitive resources. The authors distinguish between policy disclosure and policy satisfaction which allows them to have control on when a policy can be disclosed from when a policy is satisfied.

However, Yu and Winslett policies determine whether a client is authorized to be informed of the need to satisfy a given policy. While, in our case, having a separate disclosure policy $\mathcal{P}_\mathcal{D}$ allows us to have a finer-grained disclosure control over the information flow back to a client. Instead of controlling the disclosure of (entire) policies as a finest-grained unit, we are able to control the disclosure of credentials/requirements, which compose single individual policies, separately and independently from the disclosure of the policies themselves. In this case we are able to say whether a client is allowed to know for the need of any element (credential), part of a given policy, and if it is allowed then the disclosure process takes place.

11 Conclusions

In this work we proposed a framework for policy-based self-managed access control in autonomic communication. The framework is grounded in a formal model based on Datalog with the stable

model semantics. The key idea is that in an autonomic network a client may have the right credentials but may not know them and thus an autonomic communication server needs a way to avoid leaving the client stranded.

We have proposed a solution to this problem by extending classical policy-based access control models with an advanced reasoning service: *abduction*. Building on top of this service, we have presented the key interactive access control algorithm that computes on the fly the missing credentials needed for a client to get access. We extended the algorithm to cope with arbitrary access policies so that in cases of inconsistency it performs a recovery step and finds a set of excessing credentials banning the client to get a solution for the desired resource. Following that a strengthened version of the algorithm is presented that is resistant to DoS attacks. We have then proved that for cooperative and compliant clients the access control framework is complete, i.e. a client with the right credentials can always get access.

We also enriched the framework over the existing policy-based approaches for access control by introducing the difference between *disclosable* and *hidden* credentials and between *monotonic* and *well-behaved* policies. The first distinction addresses the behavior of an autonomic node allowing it to dynamically protect the privacy of his policies by specifying which credentials are hidden and which are not. This allows a server to restrict access to certain services only to selected clients. The latter distinction extends our work on a wider set of policy languages wrt the already existing approaches [5, 42, 28].

One of the advantages in our approach is that we do not pose any restrictions on partner's policies since the basic computations we perform on the policies are deduction and abduction, which do not require any specific policy structure.

Future work is in the direction of characterizing the complexity of the framweork[10], extending it to cope with mutual negotiation, and fully integrate our implementation with a privilege management infrastructure (PMI).

References

[1] APT, K. Logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Elsevier, 1990.

[2] ATLURI, V., CHUN, S. A., AND MAZZOLENI, P. A Chinese wall security model for decentralized workflow systems. In *Proceedings of the 8th ACM conference on Computer and Communications Security* (2001), ACM Press, pp. 48–57.

[3] BERTINO, E., CATANIA, B., FERRARI, E., AND PERLASCA, P. A logical framework for reasoning about access control models. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies (SACMAT)* (2001), ACM Press, pp. 41–52.

[4] BERTINO, E., FERRARI, E., AND ATLURI, V. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security (TISSEC) 2*, 1 (1999), 65–104.

---

[10]General abduction lay at the second level of the polynomial hierarchy but our problems are at the same time more specialized (e.g. credentials are occurring only positively in the rules) and more general (we have hierarchies of roles so subset or cardinality minimality does not really apply).

[5] BONATTI, P., AND SAMARATI, P. A unified framework for regulating access and information release on the web. *Journal of Computer Security 10*, 3 (2002), 241–272.

[6] CLARKE, D., ELIEN, J.-E., ELLISON, C., FREDETTE, M., MORCOS, A., AND RIVEST, R. L. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security 9*, 4 (2001), 285–322.

[7] DAMIANOU, N., DULAY, N., LUPU, E., AND SLOMAN, M. The Ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks (POLICY)* (January 2001), Springer-Verlag, pp. 18–38.

[8] DAS, S. *Deductive Databases and Logic Programming.* Addison-Wesley, Reading, MA, 1992.

[9] DE CAPITANI DI VIMERCATI, S., AND SAMARATI, P. Access control: Policies, models, and mechanism. In *Foundations of Security Analysis and Design - Tutorial Lectures*, R. Focardi and F. Gorrieri, Eds., vol. 2171 of *LNCS*. Springer-Verlag Press, 2001.

[10] ELLISON, C., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B. M., AND YLONEN, T. *SPKI Certificate Theory*, September 1999. IETF RFC 2693.

[11] FERRAIOLO, D. F., SANDHU, R., GAVRILA, S., KUHN, D. R., AND CHANDRAMOULI, R. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC) 4*, 3 (2001), 224–274.

[12] GELFOND, M., AND LIFSCHITZ, V. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference on Logic Programming (ICLP'88)* (1988), R. Kowalski and K. Bowen, Eds., MIT-Press, pp. 1070–1080.

[13] GEORGAKOPOULOS, D., HORNICK, M. F., AND SHETH, A. P. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases 3*, 2 (April 1995), 119–153.

[14] GIURI, L. Role-based access control on the web. *ACM Transactions on Information and System Security (TISSEC) 4*, 1 (2001), 37–71.

[15] JAJODIA, S., SAMARATI, P., SUBRAHMANIAN, V. S., AND BERTINO, E. A unified framework for enforcing multiple access control policies. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data* (1997), ACM Press, pp. 474–485.

[16] JOSHI, J. B. D., AREF, W. G., GHAFOOR, A., AND SPAFFORD, E. H. Security models for web-based applications. *Communications of the ACM 44*, 2 (2001), 38–44.

[17] KANG, M. H., PARK, J. S., AND FROSCHER, J. N. Access control mechanisms for inter-organizational workflow. In *Proceedings of the Sixth ACM Symposium on Access control models and technologies* (2001), ACM Press, pp. 66–74.

[18] KOSHUTANSKI, H. *Interactive Access Control for Autonomic Systems.* PhD thesis, University of Trento, Department of Information and Communication Technology, Via Sommarive 14, Povo (TN) 38100, Italy, 2005.

[19] KOSHUTANSKI, H., AND MASSACCI, F. An access control system for business processes for Web services. Tech. rep., Nordic Workshop on Secure IT Systems (NORDSEC), Gjovik University College, Norway, October 2003.

[20] KOSHUTANSKI, H., AND MASSACCI, F. A logical model for security of Web services. Tech. Rep. IIT TR-10/2003, First International Workshop on Formal Aspects in Security and Trust (FAST), Istituto di Informatica e Telematica, Pisa, Italy, September 2003. Editors: Theo Dimitrakos and Fabio Martinelli.

[21] KOSHUTANSKI, H., AND MASSACCI, F. E pluribus unum: Deduction, abduction and induction, the reasoning services for access control in autonomic communication. In *Proceedings of the 1st IFIP TC6 WG6.6 International Workshop on Autonomic Communication (WAC)* (Berlin, Germany, October 2004), vol. 3457 of *LNCS*, Springer-Verlag Press, pp. 179–190.

[22] KOSHUTANSKI, H., AND MASSACCI, F. Interactive access control for Web Services. In *Proceedings of the 19th IFIP Information Security Conference (SEC 2004)* (Toulouse, France, August 2004), Kluwer Press, pp. 151–166.

[23] KOSHUTANSKI, H., AND MASSACCI, F. An interactive trust management and negotiation scheme. In *Proceedings of the 2nd International Workshop on Formal Aspects in Security and Trust (FAST)* (Toulouse, France, August 2004), Kluwer Press, pp. 139–152.

[24] KOSHUTANSKI, H., AND MASSACCI, F. A system for interactive authorization for Business Processes for Web Services. In *Proceedings of the 4th International Conference on Web Engineering (ICWE 2004)* (Munich, Germany, July 2004), Springer-Verlag Press, pp. 521–525.

[25] KOSHUTANSKI, H., AND MASSACCI, F. Interactive credential negotiation for stateful business processes. In *Proceedings of the 3rd International Conference on Trust Management (iTrust)* (Rocquencourt, France, May 2005), vol. 3477 of *LNCS*, Springer-Verlag Press, pp. 257–273.

[26] LEONE, N., PFEIFER, G., AND ET AL. The DLV system. In *the 8th European Conference on Artificial Intelligence (JELIA)* (September 2002), vol. 2424 of *Lecture Notes in Computer Science*, Springer-Verlag Press, pp. 537–540.

[27] LI, N., GROSOF, B. N., AND FEIGENBAUM, J. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security (TISSEC) 6*, 1 (2003), 128–171.

[28] LI, N., AND MITCHELL, J. C. RT: A role-based trust-management framework. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX III)* (Los Alamitos, California, April 2003), IEEE press, pp. 201–212.

[29] LI, N., WINSBOROUGH, W. H., AND MITCHELL, J. C. Distributed credential chain discovery in trust management. *Journal of Computer Security 11*, 1 (February 2003), 35–86.

[30] LYMBEROPOULOS, L., LUPU, E., AND SLOMAN, M. An adaptive policy based framework for network services management. *Plenum Press Journal of Network and Systems Management 11*, 3 (September 2003), 277–303.

[31] NIEMELÄ, I., SIMONS, P., AND SOININEN, T. Stable model semantics of weight constraint rules. In *Proceedings of the Fifth International Conference on Logic Programming and Non-monotonic Reasoning* (December 1999), Springer-Verlag Press.

[32] PARK, J., AND SANDHU, R. Towards usage control models: beyond traditional access control. In *Seventh ACM Symposium on Access Control Models and Technologies* (2002), ACM Press, pp. 57–64.

[33] PARK, J. S., AND SANDHU, R. RBAC on the Web by smart certificates. In *Proceedings of the fourth ACM workshop on Role-based access control* (1999), ACM Press, pp. 1–9.

[34] ROSCHEISEN, M., AND WINOGRAD, T. A communication agreement framework for access/action control. In *Proceedings of the Symposium on Security and Privacy* (1996), IEEE Press, pp. 154–163.

[35] SHANAHAN, M. Prediction is deduction but explanation is abduction. In *Proceedings of IJCAI '89* (1989), Morgan Kaufmann, pp. 1055–1060.

[36] SLOMAN, M., AND LUPU, E. Policy specification for programmable networks. In *Proceedings of the First International Working Conference on Active Networks* (1999), Springer-Verlag, pp. 73–84.

[37] SMIRNOV, M. Rule-based systems security model. In *Proceedings of the Second International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security (MMM-ACNS)* (2003), Springer-Verlag Press, pp. 135–146.

[38] WEEKS, S. Understanding trust management systems. In *IEEE Symposium on Security and Privacy (SS&P)* (2001), IEEE Press.

[39] WIJESEKERA, D., AND JAJODIA, S. Policy algebras for access control the predicate case. In *Proceedings of the 9th ACM conference on Computer and Communications Security* (2002), ACM Press, pp. 171–180.

[40] XACML. eXtensible Access Control Markup Language (XACML), 2004. www.oasis-open.org/committees/xacml.

[41] YU, T., AND WINSLETT, M. A unified scheme for resource protection in automated trust negotiation. In *Proceedings of the IEEE Symposium on Security and Privacy* (May 2003), IEEE press, pp. 110–122.

[42] YU, T., WINSLETT, M., AND SEAMONS, K. E. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security (TISSEC) 6*, 1 (2003), 1–42.