Resilient Delegation Revocation with Precedence for Predecessors is NP-Complete

Marcos Cramer*, Pieter Van Hertum†, Ruben Lapauw†, Ingmar Dasseville† and Marc Denecker†
* University of Luxembourg † KU Leuven

***Abstract*—In ownership-based access control frameworks with the possibility of delegating permissions and administrative rights, chains of delegated accesses will form. There are different ways to treat these delegation chains when revoking rights, which give rise to different revocation schemes. One possibility studied in the literature is to revoke rights by issuing negative authorizations, meant to ensure that the revocation is resilient to a later reissuing of the rights, and to resolve conflicts between principals by giving precedence to predecessors, i.e. principals that come earlier in the delegation chain. However, the effects of negative authorizations have been defined differently by different authors. Having identified three definitions of this effect from the literature, the first contribution of this paper is to point out that two of these three definitions pose a security threat. However, avoiding this security threat comes at a price: We prove that with the safe definition of the effect of negative authorizations, deciding whether a principal does have access to a resource is an NP-complete decision problem. We discuss two limitations that can be imposed on an access-control system in order to reduce the complexity of the problem back to a polynomial complexity: Limiting the length of delegation chains to an integer m reduces the runtime complexity of determining access to $O(n^m)$, and requiring that principals form a hierarchy that graph-theoretically forms a rooted tree makes this decision problem solvable in quadratic runtime. Finally we discuss an approach that can mitigate the complexity problem in practice without fully getting rid of NP-completeness.**

***Index Terms*—access control, delegation, revocation, resilience, negative authorization, denial, predecessor takes precedence, complexity**

I. Introduction

In ownership-based frameworks for access control, it is common to allow principals (users or processes) to grant both permissions and administrative rights to other principals in the system. Often it is desirable to grant a principal the right to further grant permissions and administrative rights to other principals. This may lead to delegation chains starting at a *source of authority* (the owner of a resource) and passing on certain permissions to other principals in the chain [12], [14], [5], [15].

Furthermore, such frameworks commonly allow a principal to revoke a permission that she granted to another principal [10], [16], [5], [2]. Depending on the reasons for the revocation, different ways to treat the chain of principals whose permissions depended on the second principal's delegation rights can be desirable [10], [6]. For example, if one is revoking a permission given to an employee because he is moving to another position in the company, it makes sense to keep in place the permissions of principals who received their permissions from this employee; but if one is revoking a permission from a user who has abused his rights and is hence distrusted by the user who granted the permission, it makes sense to delete the permissions of principals who received their permission from this user. Any algorithm that determines which permissions to keep intact and which permissions to delete when revoking a permission is called a *revocation scheme*. Revocation schemes are usually defined in a graph-theoretical way on the graph that represents which authorizations between the principals are intact.

Hagström et al. [10] have presented a framework for classifying possible revocation schemes along three different dimensions: the extent of the revocation to other grantees (propagation), the effect on other grants to the same grantee (dominance), and the permanence of the negation of rights (resilience). This classification was based on revocation schemes that had been implemented in database management systems [9], [7], [4], [3].

The resilience dimension in Hagström et al.'s framework distinguishes revocation by removal (deletion) of positive authorizations from revocation by issuing a negative authorization which just inactivates positive authorizations. When defining which positive authorizations get inactivated by negative authorizations, Hagström et al. have not taken care to ensure that the outcome is as desired when multiple negative authorizations interact: A system implementing negative revocations as defined by Hagström et al. would in certain scenarios grant access to an access-requesting principal even though all delegation chains linking the owner of the resource to the access-requesting principal contain a principal that has issued a negative authorization to the access-requesting principal. As discussed in subsection III-D, we consider this a security threat. The first definition of negative revocation schemes that avoids this threat was provided by Cramer et al. [6].

This paper has three main contributions (the third of which consists of three parts):

- In section III, we compare different definitions of the effect of negative authorizations from the literature, show that the definitions in [10] and [1] pose a security threat, while the definition in [6] avoids this threat.
- However, there is a computational price to pay: In section IV, we prove that deciding whether a principal does have access to a resource in a system allowing for negative authorizations that behave as defined in [6] is an NP-complete decision problem, i.e. it is both in NP and NP-hard.

DOI 10.1109/CSF.2016.37

- In sections V, VI and VII, we describe three ways of solving or handling this complexity problem:
  - In section V, the complexity of the problem is reduced to polynomial complexity by limiting the length of delegation chains.
  - In section VI, the complexity of the problem is reduced to quadratic by requiring that principals form a hierarchy that graph-theoretically forms a rooted tree.
  - In section VII, we present a method that can mitigate the complexity problem in practice without fully getting rid of NP-completeness.

II. Preliminaries and related work

Hagström et al. [10] have introduced three dimensions according to which revocation schemes can be classified. These are called *propagation*, *dominance* and *resilience*:

Propagation. The decision of a principal i to revoke an authorization previously granted to a principal j may either be intended to affect only the direct recipient j or to propagate and affect all the other users in turn authorized by j. In the first case, we say that the revocation is *local*, in the second case that it is *global*.

Dominance. This dimension deals with the case when a principal losing a permission in a revocation still has permissions from other grantors. If these other grantors' revocation rights are dependent on the revoker, the revoker can dominate over these grantors and revoke the permissions from them. This is called a *strong* revocation. The revoker can also choose to make a *weak* revocation, where permissions from other grantors to a principal losing a permission are kept.

Resilience. This dimension distinguishes revocation by removal (deletion) of positive authorizations from revocation by issuing a negative authorization which just inactivates positive authorizations. In the first case another principal may grant a similar authorization to the one that had been revoked, so the effect of the revocation does not persist in time. In the second case a negative authorization will overrule any (new) positive permission given to the same principal, so its effect will remain until the negative permission is revoked. We call a revocation of the first kind a *delete* or *non-resilient* revocation, and a revocation of the second kind a *negative* or *resilient* revocation.

Since there are two possible choices along each dimension, Hagström et al.'s framework allows for eight different revocation schemes.

Hagström et al. defined their eight revocation schemes semi-formally. As is explained in detail in section 3.1 of [6], these definitions contradict some of the properties that Hagström et al. claim the revocation schemes to have. Aucher et al. [1] formalized Hagström et al.'s framework in a dynamic propositional logic, removing some of these problems in Hagström et al.'s definitions.

As explained in subsection III-D below, both Hagström et al.'s and Aucher et al.'s definition of the effect of revocations via negative authorizations pose a serious security threat. Cramer et al. [6] have proposed and formalized a refined revocation framework that builds on Hagström et al.'s framework and avoids this security threat. Among other differences, Cramer et al.'s definitions of the revocation schemes differ from Hagström et al.'s and Aucher et al.'s definitions in how interacting revocations are handled. This is discussed in detail in the next section.

A. Conflict resolution policies

Delegation frameworks that allow issuing negative authorization can bring about a state in which a conflict may arise. If a principal is granted both a positive and a negative authorization for the same object, then we say that these two authorizations *conflict* each other. A system's *conflict resolution policy* determines how to resolve such a conflict. Here is a list of possible conflict resolution policies as described by Ruan and Varadharajan [13]:

Negative-takes-precedence: If there is a conflict occurring on the authorization for some object, the negative authorizations will take precedence over the positive one.

Positive-takes-precedence: Positive authorizations from i to j take precedence over negative authorizations from k to j for all $k \neq i$. This means that a negative authorization from i to j directly inactivates only positive authorizations from i to j, and leaves other permission to j active.

Strong-and-Weak: Authorizations are categorized in two types, strong and weak. The strong authorizations always take precedence over the weak ones. Conflicts among strong authorizations are not allowed. In conflicts between weak authorizations negative ones take precedence. Note that the intended meaning of *strong* and *weak* in this policy differs from their meaning in Hagström et al.'s dominance dimension.

Time-takes-precedence: New authorizations take precedence over previously existing ones. Note that this policy will make negative authorizations non-resilient.

Predecessor-takes-precedence: If the principal i delegates (possibly transitively) some right to principal j, then authorizations issued by i to some other principal k concerning that right will take precedence over authorizations issued by j to k. In other words, the priority of subjects decreases as the privilege is delegated forward.

Note that according to Hagström et al.'s definition of a *strong* revocation mentioned above, the effect of a strong revocation is supposed to be the same as the effect that issuing a negative authorization in the context of a predecessor-takes-precedence conflict resolution policy. However, Hagström et al. do not assume the conflict resolution policy of the system to be prdecessor-takes-precedence. Instead, they consider the effect of the revocation schemes they defined in both a positive-takes-precedence and a negative-takes-precedence conflict resolution policy. They achieve the effect that a single negative authorization has in a predecessor-takes-precedence conflict resolution policy by potentially producing multiple negative authorizations in the course of a single strong revocation.

When Hagström et al.'s definition of a strong revocation is applied in the context of a negative-takes-precedence policy, the effect of a strong revocation is not as described in Hagström et al.'s definition of a strong revocation mentioned above, since with this policy a negative authorization from i to j dominates all positive authorizations granted to j, not just the ones granted by a principal dependent on i.

Cramer et al. [6] have incorporated the choice of how to resolve conflicts into the dominance dimension by extending Hagström et al.'s dominance dimension into a dimension involving three instead of just two choices:

- **Weak revocations.** These have the same intended behaviour as Hagström et al.'s weak revocations in a positive-takes-precedence policy: The principal performing the revocation only dominates over direct authorizations granted by herself, authorizations from other grantors are kept intact.
- **Predecessor-takes-precedence (p-t-p) revocations.** These have the same intended behaviour as Hagström's et al.'s strong revocations in a positive-takes-precedence policy: The principal performing the revocation dominates over other grantors' authorizations that are dependent on her.
- **Strong revocations.** These have the same intended behaviour as any of Hagström's et al.'s revocations in a negative-takes-precedence policy: The principal performing the revocation dominates over all other grantors' authorizations.

In this paper we focus on revocations that are on the domoinance dimension like Hagström et al.'s strong revocations and like Cramer et al.'s predecessor-takes-precedence revocations.

B. Revocations and denials

A revocation of a principal's rights removes rights that the principal already has. A denial of rights on the other hand can be issued even when the principal does not yet have the concerning rights, and has the effect that other principals will no longer be able to effectively grant rights to the affected principal.

Negative authorizations can function as a form of denial. When, for example, j does not yet have the rights in question and i issues a negative authorization for those rights to j, this negative authorization functions like a denial rather than like a revocation.

The work in this paper applies to negative authorizations independently of whether they are used to revoke existing rights or deny rights. We will for the rest of this paper only use the term "revocation" and not "denial", in order to be consistent with the terminology used in the papers that we extensively refer to in this paper.

III. Revocation via negative authorization

In this section we introduce the notion of a *negative authorization* and of *revocation via negative authorization*. After explaining the motivation behind negative authorizations, we define a graph-theoretic framework in subsection III-A that allows us to talk formally about the effects of negative authorizations. In subsections III-B and III-C we present two accounts from the literature of defining the effects of negative authorizations. In subsection III-D we present a serious security threat that both of these accounts face. In subsection III-E we explain how this problem has been solved by Cramer et al. [6], and briefly explain why their approach is immune to this kind of security threats.

The idea behind a negative authorization is that it does not only take away a certain permission or administrative right from a user, but also blocks the user from getting this access from another user, at least from a user dependent on the issuer of the negative authorization. For example, if Alice distrusts Bob and wants to do all that she can to ensure that Bob will not get access to a certain file, she can issue a negative authorization towards Bob, thus ensuring that even if someone whom she granted the right to further grant access rights to that file does grant Bob access right, Bob will still be blocked from accessing the file.

This paper is only about delegation revocation via negative authorizations, so we will not say anything about delegation revocation by deletion of authorizations. A revocation scheme based on issuing a negative revocation is called a *negative revocation scheme*. In Hagström et al.'s framework, there are four different negative revocation schemes, because there are four possible choices to be made along the dominance and propagation dimensions of their framework (see section II). The behaviour of weak negative authorizations is clear and does not pose any of the problems discussed below. So in our discussion, we will focus on revocation schemes that are called *strong* by Hagström et al. [10] and Aucher et al. [1] and called *predecessor-takes-precedence* (*p-t-p*) by Cramer et al. [6]. The problems discussed below apply equally to global and local revocation schemes; for the sake of simplicity, we will concentrate on global revocation schemes in this paper.

So for the rest of the paper, the only revocation scheme we consider is what Hagström et al. [10] and Aucher et al. [1] call a *Strong Global Negative* revocation, and what Cramer et al. [6] call a *P-t-p Global Resilient* revocation.

A. Graph-theoretic framework

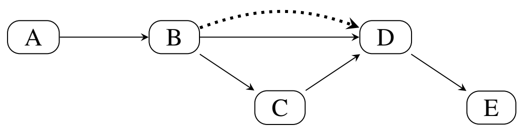
Even though this revocation scheme called Strong Global Negative in [10] and [1] and P-t-p Global Resilient in [6] is intended to be basically the same revocation scheme and does behave identically in simple scenarios, the formal definitions of this revocation scheme are somewhat different across these papers, resulting in a different behaviour especially when multiple revocations interact. In order to compare these different definitions, we describe the possible effects of negative authorizations in graph-theoretic terms. To this end, let us first define the notion of a *delegation-revocation graph*:

Definition 1: A *delegation-revocation graph* is a graph $G = (V, \mathrm{SOA}, E_+, E_-)$ consisting of a set V of vertices, also called *principals*, a distinguished vertex $\mathrm{SOA} \in V$ called the *source of authority*, a set $E_+ \subseteq V^2$ of *positive edges*, also called *positive authorizations*, and a set $E_- \subseteq V^2$ of *negative edges*, also called *negative authorizations*.

Here an example of a simple delegation-revocation graph:

Example 1: A delegation-revocation graph for a resource owned by user A

Unbroken straight edges denote positive authorizations. Dotted bent edges denote negative authorizations.



The delegation-revocation graph keeps track of which positive and negative authorization concerning a certain resource have been issued between the principals in the system. For example, the delegation-revocation graph 1 expresses that B has issued both a positive and a negative authorization towards D, but only a positive authorization towards C. The source of authority is the owner of the resource in question: All permissions emanate from the source of authority, i.e. a principal can have a permission only if there is a *delegation chain* from the source of authority to that user.

Since in this paper we are only considering revocation schemes for which the temporal order of the issuing of the authorizations is irrelevant, the authorization in a delegation-revocation graph do not have timestamps.

Hagström et al. [10], Aucher et al. [1] and Cramer et al. [6] all distinguish between positive authorizations that only grant access and positive authorizations that additionally grant delegation rights. In a delegation chain starting at the source of authority, all but the last positive authorization along the chain have to grant delegation rights. The issues discussed in this paper arise from the granting of delegation rights and are independent of whether the system allows users to grant access rights without granting delegation rights. In order to keep the exposition simple, we consider all positive authorizations to grant delegation rights. This is why we have only one kind of positive authorizations in our definition of a delegation-revocation graph. The results presented in this paper can easily be extended to systems that do have positive authorizations that only grant access rights additionally to the positive authorizations that also grant delegation rights.

Given that we have only one kind of positive authorizations, the definition of a *delegation chain* is very simple:

Definition 2: In a delegation-revocation graph, a *delegation chain* is a chain of positive authorizations starting at the source of authority.

B. Hagström et al.'s Strong Global Negative revocation

Hagström et al. [10] define the effect of Strong Global Negative revocations using the notion of a principal being *independent* of another principal i, i.e. possessing delegation rights in a way that is not dependent on i:

Definition 3: A principal k is *independent* of a principal i iff there is a delegation chain ending in k that does not pass through i.

Note in particular that a principal k is never independent of k.

Hagström et al. [10] use this notion of independence to define the effect of a Strong Global Negative revocation by defining which positive authorizations are inactivated by a negative authorization. There are two ways that a positive authorization can be inactivated in their definition:

1) A negative authorization from i to j inactivates a positive authorization from k to j if k is not *independent* of i.
2) Furthermore a positive authorization issued by a principal k is inactivated if there is no active positive authorization granting k delegation rights.

Case 1 takes care that the principal targeted by the Strong Global Negative revocation loses her delegation rights if her rights are not supported in a way independent of the principal performing the revocation. Case 2 takes care of propagating forward the effect of the revocation (given that it is a global rather than a local revocation).

For example, in the delegation-revocation graph in Example 1, the negative authorization from B to D inactivates (based on case 1) the positive authorization from B to D, because B is not independent of B. Furthermore it inactivates (also based on case 1) the positive authorization from C to D, because C is not independent of B: The only delegation chain ending in C goes through B. Once the positive authorizations from B to D and from C to D are inactivated, the authorization from D to E is also inactivated (based on case 2), as there is no active positive authorization left that grants D delegation rights.

C. Aucher et al.'s Strong Global Negative revocation

Definition 3 is problematic, because it does not correctly formalize the intended meaning of *independent*: Suppose there is precisely one delegation chain ending in k and not passing through i, but this delegation chain contains inactive authorizations. In that case, this delegation chain does not in any way support the rights of k. But according to definition 3, this delegation chain would ensure that k is independent of i. The problem is, of course, that definition 3 does not specify that the delegation chain should be active, i.e. not contain inactive authorizations.

In their formalization of Hagström et al.'s framework in a dynamic propositional logic, Aucher et al. [1] corrected this problem of Hagström et al.'s definition. However, instead of defining the effect a negative authorization by defining which positive authorizations it inactivates, they add auxiliary negative authorizations to get the same result concerning which users lose their access and delegation rights:

They define a Strong Global Negative revocation from i to j to consist of issuing a negative authorization from i to j and additionally adding auxiliary negative authorizations from k to j for for every k that is not independent of i. They define k to be independent of i iff there is a *rooted delegation chain* from the SOA to i that does not contain i. A *rooted delegation chain* is defined to be a delegation chain such that for any two consecutive principals p, q along this delegation chain, there is no negative authorization from p to q. A principal i is defined to have access and delegation rights iff there is a rooted delegation chain from the SOA to i.

The fact that the definition of independence refers only to rooted delegation chains rather than to any delegation chains removes the problem with Hagström et al.'s definition of independence.

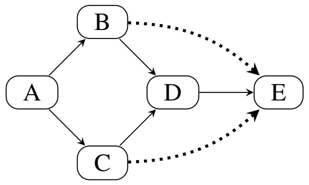
For example, the issuing of the negative authorization from B to D in the delegation-revocation graph 1 would in the case of a Strong Global Negative revocation entail adding an auxiliary negative authorization from C to D (because C is not independent of B). Hence none of the two delegation chains from A to D are rooted delegation chains, so D does no longer have delegation rights, as expected. Similarly, none of the two delegation chains from A to E are rooted delegation chains, so also E does no longer have delegation rights.

D. A security threat

Even though Aucher et al. removed one problem in Hagström et al.'s definition of the effect of Strong Global Negative revocations, there remains another problem that is present in both Aucher et al.'s and Hagström et al.'s framework and that poses a security threat. This subsection describes this security threat.

Consider the following delegation-revocation graph:

Example 2:



The only two delegation chains from the SOA to E are ABDE and ACDE. Both of these delegation chains contain a principal that has issued a negative authorization towards E. By issuing a negative authorization to E, B and C have expressed distrust in E and intended to avoid that E gets access and delegation rights through a user dependent on them. Hence E should not have access and delegation rights in this scenario.

But according to the effect of negative authorizations as defined by Hagström et al. [10] and Aucher et al. [1], E would actually have access and delegation rights in this situation: D is independent of B, because there is a delegation chain ACD not going through B; and D is independent of C, because there is a delegation chain ABD not going through C. So neither the negative authorization from B to E nor the negative authorization from C to E can inactivate the positive authorization from D to E (in Hagström et al.'s account) or lead to the addition of an auxiliary negative authorization from D to E (in Aucher et al.'s account).

The problem is that in their definitions of the effect of a negative authorization, both Hagström et al. and Aucher et al. only accounted for the desired effect of a single negative authorization, without taking into account what the desired outcome should be when multiple negative authorizations interact. In example 2, the two negative authorizations work together to discredit user E; neither of the two negative authorizations by itself should have any negative effect on E.

This problem poses a security threat because it allows principals to have access and delegation rights that they should not have. For instance, in example 2, Hagström et al.'s and Aucher et al.'s frameworks would grant E access and delegation rights even though E should not be granted these rights.

E. Solving the problem

One way to explain the problem is to point out that both Hagström et al. and Aucher et al. have limited their notion of *independence* to independence from a single user. But one can also define the notion of a user being independent from a set of users:

Definition 4: A principal i is defined to be independent of a set S of principals iff there is an active delegation chain (i.e. a delegation chain all of whose positive authorizations are active) from the SOA to i that does not include any principal in S.

Now one can replace Hagström et al.'s first way of inactivating positive authorizations by the following definition of inactivation: When there is a set S of principals such that from every principal $i \in S$ there is a negative authorization towards j, and k is not independent of S, then a positive authorization from k to j gets inactivated.

This desired effect of multiple interacting negative authorizations can also be phrased in more simple terms, without reference to the notions of independence and inactivation: When all delegation chains linking the source of authority to j contain a principal that has issued a negative authorization to j, then j should not have access or delegation rights.

This way of defining the effect of multiple interacting negative authorizations was introduced by Cramer et al. [6]. Formally, the effect can be defined by postulating that a principal i should have access and delegation rights iff there is a *good delegation chain* from the SOA to i, where a *good delegation chain* is defined as follows:

Definition 5: Let $G = (V, \text{SOA}, E_+, E_-)$ be a delegation-revocation graph. A *good delegation chain* in G is a sequence of vertices $v_0, \dots, v_n$ such that $v_0 = \text{SOA}$, $(v_i, v_{i+1}) \in E_+$ for $0 \leq i < n$, and $(v_i, v_j) \notin E_-$ for any $0 \leq i \leq j \leq n$.

In other words, a good delegation chain is a delegation chain with the property that no principal in the chain has issued a negative authorization towards a principal later in the delegation chain.

In order to explicitly distinguish Cramer et al.'s definition of when a principal has access right from Hagström et al.'s and Aucher et al.'s definitions, we will say that a principal has *safe access right* if she has access right according to Cramer et al.' definition:

Definition 6: Let $G = (V, \text{SOA}, E_+, E_-)$ be a delegation-revocation graph, and let $p \in V$ be a principal. We say that p has *safe access right* iff there is a good delegation chain in G that ends in p.

In Example 2, the delegation chain ABDE is not a good delegation chain, because there is a negative authorization from B to E, and B occurs in ABDE before E. Similarly, ACDE is not a good delegation chain because of the negative authorization from C to E. So there is no good delegation chain from the SOA to E, i.e. E does not have safe access right.

So Cramer et al. [6] have removed the security threat that we have identified in subsection III-D. But can we be sure that no similar security threat is still faced by Cramer et al.'s definition? In order to clear up such doubts, Cramer et al. [6] have developed a formal framework based on the notions of trust and distrust for reasoning about the reasons for delegating and revoking. This formal framework allowed them to formally define the desired behaviour of interacting revocations. This desired behaviour includes avoiding security threats like the one identified in subsection III-D, but generalizes the eschewal of such threats to arbitrary complex scenarios. They established that the above definition of the effect of negative authorizations has this desired behaviour, while the definitions by Hagström et al. and Aucher et al. do not have this desired behaviour.

The downside of Cramer et al.'s definition of the effect of negative authorizations is that it makes the decisions problem of deciding whether a principal has access or delegation right NP-complete, as we will see in the following section. In the case of Hagström et al.'s and Aucher et al.'s definition, on the other hand, there is an algorithm with runtime quadratic in the number of principals that decides whether a principal has access or delegation rights.

IV. NP-COMPLETENESS

In the previous section we saw that the revocation frameworks by Hagström et al. [10] and Aucher et al. [1] have posed a security threat, and that this problem has been solved by Cramer et al. [6] by giving a new definition of the effect of negative authorizations. In this section we show that Cramer et al.'s solution comes at a price: Whereas in Hagström et al.'s and Aucher et al.'s framework it could be determined in quadratic runtime whether a principal has access or delegation right, in Cramer et al.'s framework the decisions problem of deciding whether a principal p has safe access right is NP-complete.

For this we need to show that this decision problem is both in NP and NP-hard. It can be easily seen to be in NP, because a good delegation chain from the SOA to p is a witness for this decision problem: p has safe access right iff there is a good delegation chain from the SOA to p, and determining whether a given sequence of principals is a good delegation chain can clearly be checked in polynomial time (namely in quadratic time).

In order to prove that this decision problem is NP-hard, we reduce 3-SAT [11] to this decision problem. Let $a_1, \dots, a_n$ be propositional variables, and let $C_1 \wedge \dots \wedge C_m$ be a 3-SAT problem in $a_1, \dots, a_n$, i.e. each C_i is a clause of the form $l_1^i \vee l_2^i \vee l_3^i$, where each l_j^i is a literal of the form a_k or $\neg a_k$. We need to construct a delegation-revocation graph such that for some specific principal p in this graph, p has safe access right iff $C_1 \wedge \dots \wedge C_m$ is satisfiable. We describe the construction of this delegation-revocation graph for the general case, and illustrate it in figure 3 with the delegation-revocation graph corresponding to the 3-SAT problem $(a_1 \vee a_2 \vee a_3) \wedge (\neg a_1 \vee a_2 \vee \neg a_3)$. It can be easily seen that the order of the delegation-revocation graph is linear in the size of the 3-SAT problem (more precisely, if n denotes the number of clauses of the 3-SAT problem, the order of the delegation-revocation graph is always less than or equal to $10n + 2$).

The delegation-revocation graph has the following principals:

- SOA
- a_i and $\neg a_i$ for $1 \leq i \leq n$
- l_j^i for $1 \leq i \leq m$ and $1 \leq j \leq 3$
- SAT_i for $0 \leq i \leq m$

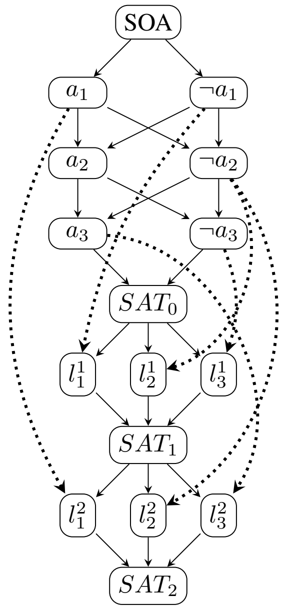
The graph has the following positive edges:

- From SOA to a_1 and to $\neg a_1$
- From a_i to a_{i+1} and $\neg a_{i+1}$, for $1 \leq i < n$
- From $\neg a_i$ to a_{i+1} and $\neg a_{i+1}$, for $1 \leq i < n$
- From a_n and $\neg a_n$ to SAT_0.
- From SAT_i to l_j^{i+1} for $0 \leq i < m$ and $1 \leq j \leq 3$
- From l_j^i to SAT_i for $1 \leq i \leq m$ and $1 \leq j \leq 3$

The graph has the following negative edges:

- From a_k to any l_j^i that is of the form $\neg a_k$
- From $\neg a_k$ to any l_j^i that is of the form a_k

Example 3: The delegation-revocation graph corresponding to the 3-SAT problem $(a_1 \vee a_2 \vee a_3) \wedge (\neg a_1 \vee a_2 \vee \neg a_3)$



The following lemma establishes the desired result that determining safe access right for principal SAT_m in the constructed delegation-revocation graph amounts to solving the 3-SAT problem for $C_1 \wedge \dots \wedge C_m$:

Lemma 1: SAT_m has safe access right iff $C_1 \wedge \dots \wedge C_m$ is satisfiable.

Proof: First note that every path from SOA to SAT_0 corresponds to an assignment of truth values to $a_1, \dots, a_n$: For each $a \leq i \leq n$, each such path goes through precisely one of a_i and $\neg a_i$, and we assign a_i the truth value *True* in the corresponding truth value assignment iff the path goes through a_i.

Every path from SOA to SAT_m consists of a path from SOA to SAT_0 followed by a path from SAT_0 to SAT_m. The subpath from SAT_0 to SAT_m goes through precisely one of l_1^i, l_2^i and l_3^i for each $1 \leq i \leq m$. The path from SOA to SAT_m can only be a good delegation chain if for every $1 \leq i \leq m$, the literal l_j^i through which the path goes is assigned the truth value T under the truth value assignment corresponding to the subpath from SOA to SAT_0.

So SAT_m has safe access right

- iff there is a good delegation chain from SOA to SAT_m
- iff there is a path from SOA to SAT_0 and a path from SAT_0 to SAT_m such that for every $1 \leq i \leq m$, the literal l_j^i through which the from SAT_0 to SAT_m goes is assigned the truth value T under the truth value assignment corresponding to the subpath from SOA to SAT_0
- iff there is an assignment of truth values to $a_1, \ldots, a_n$ and a choice of $l_j^i \in \{l_1^i, l_2^i, l_3^i\}$ for every $1 \leq i \leq m$ such that for every $1 \leq i \leq m$, l_j^i is true under this truth value assignment
- iff $C_1 \wedge \cdots \wedge C_n$ is satisfiable.

■

Hence we have established the following theorem:

Theorem 1: The decision problem of determining whether a principal p in a delegation-revocation graph has safe access right is NP-complete with respect to the order of the delegation-revocation graph.

V. Bounded delegation depth

Given that deciding safe access right in an access-control framework with no limitations on delegations is NP-complete, it is of interest to study sensible limitations that can be imposed on an access-control framework in order to reduce the complexity of this decision problem to polynomial time. In this section we consider the possibility of limiting the maximum *delegation depth*, i.e. the maximum length of delegation chain, by a constant integer.

Multiple access control frameworks allow for limiting the delegation depth [16], [12]. However, in these frameworks the maximum delegation chain can be freely chosen by the source of authority, and there is no upper bound for the integers that the source of authority may choose for this purpose.

The limitation that we consider in this section is different in nature: We suppose that a fixed integer m is defined in the access control policy to be the universal maximum delegation depth, and no source of authority can allow for deeper delegation of the rights that he or she delegate to others. We call this fixed integer m the *bound on delegation depth*. The following theorem establishes that if there is such a bound on delegation depth, then the complexity of deciding safe access right is polynomial in the number of principals:

Theorem 2: Let m be the bound on the length of delegation chains. Then the runtime complexity of deciding whether a principal has safe access right is at most $O(n^m)$, where n denotes the number of principals.

Proof: Assuming that the length of delegation chains may not be more than m, Algorithm 1 decides whether a principal p has safe access right. It does this by checking for every sequence of principals of length at most m whether it is a good delegation chain from SOA to p: After selecting such a sequence σ, the value of the Boolean b' is set to True (line 3). If some reason is found for concluding that σ is not a good delegation chain from SOA to p, the value of b' is changed to False (lines 4-13): This can happen either because the sequence does not start at SOA or does not end at p (lines 4-5), because some principal in the sequence has not granted a positive authorization to the next principal in the sequence (lines 8-9), or because there is a negative authorization from one principal to a later principal in the sequence (11-13). The breaks in lines 9, 13 and 15 ensure that once it has been established that a sequence is not a good delegation chain, the algorithm immediately stops considering this sequence. As soon as a good delegation chain has been found, access right is granted and the algorithm stops (lines 16-17).

Algorithm 1 Determining safe access with bounded delegation chain length

```
Input: delegation-revocation graph $G = (V, \text{SOA}, E_+, E_-)$,
    principal $p \in V$
Output: a Boolean $b$ stating whether $p$ is granted safe access
    right or not
 1: $b \leftarrow$ False
 2: for $\sigma$ a sequence of principals in $G$ with length($\sigma$) $\leq m$
    do
 3:   $b' \leftarrow$ True
 4:   if $\sigma[0] \neq$ SOA or $\sigma[\text{length}(\sigma) - 1] \neq p$ then
 5:     $b' \leftarrow$ False
 6:   else
 7:     for $i \in \{0, \ldots, \text{length}(\sigma) - 2\}$ do
 8:       if $(\sigma[i], \sigma[i+1]) \notin E_+$ then
 9:         $b' \leftarrow$ False, break
10:       if $b' =$ True then
11:         for $j \in \{i, \ldots, \text{length}(\sigma) - 1\}$ do
12:           if $(\sigma[i], \sigma[j]) \in E_-$ then
13:             $b' \leftarrow$ False, break
14:       if $b' =$ False then
15:         break
16:   if $b' =$ True then
17:     $b \leftarrow$ True, break
```

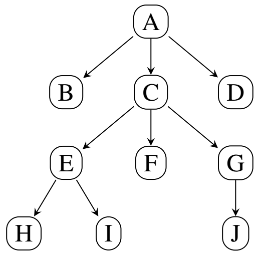
Now we show that the algorithm's runtime is at most $k \cdot n^m$ for some constant k. First note that there are less than $m \cdot n^m$ sequences of principals of length at most m. Hence it is enough to show that the time needed for the execution of lines 3-17 (checking whether a given sequence is a good delegation chain) has a constant upper bound. This follows from the fact that both in the **for**-loop starting in line 7 and in the **for**-loop starting in line 11, the number of iterations is bounded by the constant m. ■

VI. Delegation revocation in a hierarchical setting

In this section we consider another limitation that can be imposed onto the access-control framework in order to avoid the NP-completeness of deciding safe access right. Roughly speaking, the limitation is that principals form a hierarchy that graph-theoretically forms a rooted tree, and that all delegation and revocation has to respect this hierarchy. After making this limitation more precise, we show that it reduces the complexity of deciding safe access right for principal p to runtime quadratic in the hierarchical depth of of principal p (this depth is at most the number of principals in the system, but usually much less).

Let us first motivate the limitation that we impose in this section: Suppose that a company wants to use an access-control framework that allows for delegation. Suppose for simplicity that every principal in the company's computer system is a human user. Furthermore, suppose that the employees of the company form a hierarchy in which everyone apart from the company's CEO has precisely one direct boss. So the employees of the company form a hierarchy like the one depicted in example 4 that has the graph-theoretical structure of a *rooted tree*. In other words, when depicting the employees as vertices and the relation between an employee and her direct subordinate by a directed edge, the result is a directed graph with a distinguished vertex, the *root*, corresponding to the company's CEO, such that there is precisely one path from the root to any other vertex in the graph.

Example 4: Example of a hierarchy with the graph-theoretical structure of a rooted tree with root A



In example 4, A is the direct boss of B, C and D; C is the direct boss of E, F and G; E is the direct boss of H and I; and G is the direct boss of J.

In practice, there will also be principals that do not correspond to employees of the company but to processes in the company's computer network. Nevertheless, it might still be possible to impose such a tree-like hierarchical structure onto the principals.

The idea now is to limit delegation, i.e. the issuing of positive authorizations, and revocation, i.e. the issuing of negative authorizations, in the following way: A principal i may only issue a positive authorization to principal j if i is directly above j in the hierarchy. For negative authorizations the restriction is less strict: A principal i may only issue a negative authorization to principal j if i is (directly or indirectly) above j in the hierarchy , i.e. if there is a path from i to j in the directed graph that represents the hierarchy. Thus in example 4, A can issue positive authorizations to B, C and D, and negative authorizations to everyone, whereas, for example, C can issue positive authorizations to E, F and G, and negative authorizations to E, F, G, H, I and J.

The distinction between the more strict limitation on positive authorizations and the less strict limitation on negative authorizations can be motivated as follows: In order to get some permission, one's direct boss should consent. But for taking away a permission there should be more flexibility, because for avoiding security threats it is important to react quickly when a principal is identified as not trustworthy: If only the direct boss could take away someone's permission, this would cause additional delay, for example if the direct boss of the problematic principal is currently not available. By allowing the boss's boss, the boss's boss's boss etc. to also take away a principal's permission, a quick reaction to the identification of a principal as harmful is more likely.

We are not making any assumptions about whether the tree-like hierarchical structure is constant or can be modified over time. We don't even have to assume that the same hierarchical structure is used for all rights: There could in principle be different hierarchical structures corresponding to different delegatable rights.

When we focus on the authorizations issued for one particular access right, the only part of the hierarchy tree that is relevant is the subtree of the hierarchy tree that consists of the source of authority for that access right and all principals (directly or indirectly) below this source of authority. This subtree is again a rooted tree, now with the source of authority at its root. The rest of the hierarchical tree is not relevant for determining this access right and will be ignored for the rest of the discussion in this section. So any reference to a rooted tree from now onwards will concern this subtree of the original hierarchical tree.

The following notion of a *hierarchical delegation-revocation graph* formalizes the situation in which we have to determine access given the hierarchical limitation on issuing authorizations discussed above:

Definition 7: A *hierarchical delegation-revocation graph* is a graph $G = (V, \text{SOA}, H, E_+, E_-)$ consisting of a set V of vertices, also called *principals*, a distinguished vertex $\text{SOA} \in V$ called the *source of authority*, a set $H \subseteq V^2$ of *hierarchy edges*, a set $E_+ \subseteq V^2$ of *positive edges*, also called *positive authorizations*, and a set $E_- \subseteq V^2$ of *negative edges*, also called *negative authorizations*, satisfying the following properties:

1) For every principal $p \in G$ there is precisely one path of hierarchy edges connecting SOA to p.
2) $E_+ \subseteq H$.
3) Whenever $(p_1, p_2) \in E_-$, there is a path of hierarchy edges from p_1 to p_2.

This definition extends definition 1 of a *delegation-revocation graph* from subsection III-A by adding a set H of hierarchy edges. Property 1 ensures that the principals and hierarchy edges form a rooted tree with the SOA as root. Properties 2 and 3 ensure that authorizations have only been issued in accordance with the limitation defined above: By property 2, a positive authorization from p_1 to p_2 can only have been issued if p_1 is directly above p_2 in the hierarchy, i.e. if $(p_1, p_2) \in H$. By property 3, a negative authorization from p_1 to p_2 can only have been issued if p_1 is (directly or indirectly) above p_2 in the hierarchy, i.e. if there is a path of hierarchy edges from p_1 to p_2.

The definitions for good delegation chains and safe access in hierarchical delegation-revocation graphs are the same as the corresponding definitions for non-hierarchical delegation-revocation graphs in subsection III-E above:

Definition 8: Let $G = (V, \text{SOA}, H, E_+, E_-)$ be a hierarchical delegation- revocation graph. A *good delegation chain* in

G is a sequence of vertices $v_0, \dots, v_n$ such that $v_0 = \text{SOA}$, $(v_i, v_{i+1}) \in E_+$ for $0 \leq i < n$, and $(v_i, v_j) \notin E_-$ for any $0 \leq i \leq j \leq n$.

Definition 9: Let $G = (V, \text{SOA}, H, E_+, E_-)$ be a hierarchical delegation- revocation graph, and let $p \in V$ be a principal. We say that p has *safe access right* iff there is a good delegation chain in G that ends in p.

In order to state the theorem about the complexity of determining safe access right, we need the notion of the *hierarchical depth* of a principal:

Definition 10: Let $G = (V, \text{SOA}, H, E_+, E_-)$ be a hierarchical delegation- revocation graph, and let $p \in V$ be a principal. The *hierarchical depth* of p in G is the length of the unique path of hierarchy edges from SOA to p.

The following theorem establishes that the runtime complexity of determining safe access right of a principal p is quadratic in the hierarchical depth of principal p:

Theorem 3: Let $G = (V, \text{SOA}, H, E_+, E_-)$ be a hierarchical delegation-revocation graph, and let $p \in V$ be a principal. Let n denote the hierarchical depth of p in G. Then the runtime complexity of deciding whether a principal has safe access right is at most $O(n^2)$.

Proof: Algorithm 2 decides whether principal p has safe access right. It does this by looking at the unique path $[p_1, \dots, p_n]$ from SOA to p (so $p_1 = \text{SOA}$ and $p_n = p$) and determining whether it is a good delegation chain (there cannot be any other good delegation chain by the restrictions imposed on hierarchical delegation-revocation graphs). For this, two properties have to be satisfied:

- Positive authorizations have to be in place along the whole path $[p_1, \dots, p_n]$, i.e. $(p_i, p_{i+1}) \in E_+$ for $1 \leq i \leq n$: This is checked in lines 3-5 of Algorithm 2. Note that the algorithm actually checks whether this property fails for some i, setting the Boolean b to False and halting if it fails for some i.
- No negative authorizations may be in place from an earlier to a later principal in the path $[p_1, \dots, p_n]$, i.e. for no $1 \leq i \leq j \leq n$, $(p_i, p_j) \in E_-$: This is checked in lines 7-10 of Algorithm 2. The algorithm searches for such a negative authorization (p_i, p_j), and if one is found, b is set to True and the algorithm halts.

Algorithm 2 clearly has runtime at most quadratic in the hierarchical depth n of p: Finding the unique path from SOA to p (line 2) is actually linear in n. So is the procedure in lines 3-5. The time needed for executing lines 7-10 is at most $O(n^2)$ given that both the **for**-loop starting in line 7 and the **for**-loop starting in line 8 have at most n iterations. ■

VII. A PRACTICAL APPROACH TO MITIGATING THE PROBLEM

The limitations discussed in sections V and VI can be used to significantly reduce the computational cost of determining safe access right. However, depending on the institutional and technical setting in which the access control framework is to be used, these limitations may or may not be desirable. If they are not desirable, other approaches to mitigating the problem need to be adopted. In this section we therefore consider a further approach to mitigate the problem of the computational cost of deciding safe access right, which does not remove NP-completeness completely, but ensures that it can only rarely cause significant delays.

Algorithm 2 Determining safe access in a hierarchical delegation-revocation graph

```
Input: hierarchical delegation-revocation graph
    $G = (V, \text{SOA}, H, E_+, E_-)$, principal $p \in V$
Output: a Boolean $b$ stating whether $p$ is granted safe access
    right or not
 1: $b \leftarrow$ True
 2: $[p_1, \dots, p_n] \leftarrow$ the unique path from SOA to $p$
 3: for $1 \leq i \leq n$ do
 4:     if $(p_i, p_{i+1}) \notin E_+$ then
 5:         $b \leftarrow$ False, break
 6: if $b =$ True then
 7:     for $1 \leq i \leq n$ do
 8:         for $i \leq j \leq n$ do
 9:             if $(p_i, p_j) \in E_-$ then
10:                 $b \leftarrow$ False, break
```

This approach is motivated by the idea that there should normally be objective reasons for issuing a negative authorization. If Alice issues a negative authorization towards Bob, this means that Alice distrusts Bob to use the access right in question. Alice should of course have some reason for this distrust. If this reason is objective rather than subjective, then Alice should be able to convince the source of authority of the distrustworthiness of Bob. But if the SOA can be made to distrust Bob, then the SOA herself should issue a negative authorization towards Bob. And when there is a negative authorization from the SOA to Bob, Bob can under no circumstances get access right, so the negative authorization from Alice to Bob is no longer required.

In other words, a negative authorization from i to j should under normal circumstances be replaced by a negative authorization from the SOA to j.

At this point an attentive reader may ask why principals other than the SOA should be allowed to issue negative authorizations in the first place. The reason is very simple: Once Alice finds out something about Bob that suggests Bob is to be distrusted, it is important for Alice to ensure quickly that Bob does not use his access right obtained through Alice for malicious action. If only the SOA could issue negative authorizations, Alice would first have to convince the SOA of the distrustworthiness of Bob, which would give Bob additional time for performing something malicious.

This suggests that the following approach to negative authorization may be taken in practice: Principals other than the SOA can issue negative authorizations in order to ensure quick action upon the discovery of a distrustworthy principal. When issuing a negative authorization, a principal should report to the SOA the reasons for issuing this negative authorization. The SOA then decides whether the negative authorization is justified or not. If yes, the SOA herself issues a negative authorization towards the distrusted principal; this makes the original negative authorization obsolete, so that it can be removed. If not, the original authorization should also be removed. This means that under normal circumstances, negative authorizations issued by principals other than the SOA only exist for short moments of time, and most of the time the only negative authorizations in place are issued by the SOA.

This motivates the following definitions: We say that the delegation-revocation graph is in a *stable state* when all negative authorizations in place are issued by the SOA; else we say it is in an *unstable state*.

A further idea behind our approach is that access requests should in general be treated quickly, so it is not desirable to have the whole computational burden of finding good delegation chains being performed at runtime in response to an access request. Instead, this computational burden can be partly tackled offline independently of specific access requests.

The method described for the rest of this section keeps track of access rights and of some good delegation chains witnessing these access rights offline, updating the information whenever a new authorization is added. In the ideal case, the access right information is already updated at the moment an access request is received by the reference monitor, so that access can be granted or denied quickly. The method works efficiently at stable states, so that access requests occurring while the delegation-revocation graph is in a stable state can always be treated quickly. At an unstable state, the method may face a computationally expensive subroutine, which can lead the system to be unsure as to the access right status of some principals. If an access request from such a principal arrives at such a moment, significant delays may occur.

First we describe how the method works when the delegation-revocation graph constantly stays in a stable state. For every stable state the delegation-revocation graph is in, the method determines a set Access of principals that currently have access right, and a function WitnessingChain that assigns one good delegation chain to each principal in Access in such a way that for every $p \in \text{Access}$, the chain $\text{WitnessingChain}(p)$ witnesses the access right of principal p. At the beginning, there are no authorizations and only the SOA has access right, so the initial value of Access is $\{\text{SOA}\}$, and the function WitnessingChain is initially defined only at on the domain $\{\text{SOA}\}$ by setting the value of WitnessingChain(SOA) to the empty chain.

When a positive authorization from i to j is added to the delegation-revocation graph, the method checks whether i and j are in Access. If $i \in \text{Access}$ but $j \notin \text{Access}$, the sets Access and WitnessingChain are updated as follows:

- Access is set to $\text{Access} \cup \{j\}$.
- $\text{WitnessingChain}(j)$ is set to be $\text{WitnessingChain}(i) + j$, i.e. the chain resulting from extending the chain $\text{WitnessingChain}(i)$ by principal j.

If a negative authorization from SOA to j is added to the delegation-revocation graph, the method calls Algorithm 3 as a subroutine. Algorithm 3 updates the set Access and the function WitnessingChain by first determining (in lines 1-4) which chains in the co-domain of WitnessingChain are no longer good delegation chains because of the new negative authorization, and then (in lines 6-11) trying to find alternative good delegation chains that may substitute the invalidated delegation chains. GoodChainMissing is the set of principals p for which good delegation chains are missing after the new negative authorization from SOA to j invalidated the delegation chain witnessing the access right of p. After access has been temporarily removed in line 5 from every principal whose delegation chain has been invalidated, the method defined in lines 6-11 for finding alternative good delegation chains for these principals works by iteratively giving back access right to all principals that have received a positive authorization from someone with access right and that have not received a negative authorization from the SOA. After iterating this procedure |GoodChainMissing| times (i.e. as often as the number of principals that may need to be given back their access right), every principal with safe access right is guaranteed to be in Access.

Algorithm 3 Updating the set Access and the function WitnessingChain when the SOA issues a negative authorization towards j

```
Input: delegation-revocation graph $G = (V, \text{SOA}, E_+, E_-)$,
    principal $j \in V$, Access, WitnessingChain
Output: updated values for Access and WitnessingChain
 1: GoodChainMissing $\leftarrow \{\}$
 2: for $C$ in the co-domain of WitnessingChain do
 3:   if principal $j$ occurs in the chain $C$ then
 4:     GoodChainMissing $\leftarrow$ GoodChainMissing $\cup$
        $\{$the last element of $C\}$
 5: Access $\leftarrow$ Access $\setminus$ GoodChainMissing
 6: for $1 \leq i \leq |\text{GoodChainMissing}|$ do
 7:   for $p \in$ Access do
 8:     for $p' \in$ GoodChainMIssing do
 9:       if $(p, p') \in E_+$ and $(\text{SOA}, p') \notin E_-$ then
10:         Access $\leftarrow$ Access $\cup \{p'\}$
11:         WitnessingChain$(p')$ $\leftarrow$ WitnessingChain$(p)$ +
            $p'$
```

It can easily be seen that the runtime of Algorithm 3 is in the worst case cubic in the number of principals (because of the three nested **for**-loops in lines 6-11).

In the case of these modifications that leave the delegation-revocation graph in a stable state, the computations required for updating Access and WitnessingChain are reasonably fast, so that we can assume that these updates are always computed before any access request made after the modification to the graph is handled.

But when the delegation-revocation graph becomes unstable because of the addition of a negative authorization from i to j for $i \neq \text{SOA}$, the procedure of updating Access and WitnessingChain is NP-complete. We refrain from specifying a particular procedure for updating Access and WitnessingChain in this case; this update should be done using some method oriented at solving NP-hard problems, like a state-of-the-art SAT-solver [8]. Note that this update procedure may take a considerable amount of time and an access request may be made during this time. In this case, priority should be given to determining the access right of the requester before determining the access right of other principals. Of course, determining the access right of the requester is also NP-complete, so if the delegation-revocation graph is big and complex and the SAT-solver happens not to find a good delegation graph in a reasonable amount of time, the requester may have to wait until the delegation-revocation graph returns to a stable state before getting a response to his access request.

When the delegation-revocation graph returns to a stable state from an unstable state, the values of the set Access and the function WitnessingChain are determined based on their values at the last stable state and the changes that have been made to the delegation-revocation graph with respect to this last stable state.

VIII. Conclusion

In an access-control framework with the possibility to delegate permissions, it is desirable to have a possibility to revoke delegated permissions. In the literature, multiple revocation schemes have been studied and compared. One class of revocation schemes are *negative revocation schemes*, i.e. revocation schemes that function by issuing a negative authorization. In section III, we have compared three different ways that have been proposed to define the effect of negative authorizations. In subsection III-D, we have pointed out that two of these three definitions pose a security threat by not properly defining the effect of multiple interacting negative authorizations. However, the definition from Cramer et al. [6] that avoids this threat comes at a price: It makes determining access right an NP-complete problem, as we have shown in section IV.

Given that the other two definitions of the effect of negative authorizations pose a security threat, it is certainly not a good idea to avoid this complexity problem by using these definitions instead of the safe definition by Cramer et al. [6]. Instead, we have considered three possible ways of solving or handling this complexity problem:

- In section V we established that bounding delegation depth to an integer m reduces runtime complexity to a polynomial of degree m.
- In section VI we have shown that a limitation in delegation and revocation based on a hierarchical structure of the principals makes the runtime complexity of determining access for a principal p quadratic in the hierarchical depth of p.
- In section VII we have discussed an approach to mitigate the problem of computational cost in praxis, based on the idea that negative authorizations issued by principals other than the source of authority should only be considered temporary measures. We have defined a method that keeps track of who has access right offline, i.e. independently of access requests. The updates defined in this method take time at most cubic in the number of principals when the negative authorizations in place are all issued by the source of authority, but significant delays may occur when other principals issue negative authorizations that are not quickly replaced by negative authorizations issued by the source of authority.

References

[1] Aucher, G., Barker, S., Boella, G., Genovese, V., van der Torre, L.: Dynamics in Delegation and Revocation Schemes: A Logical Approach. In: Li, Y. (ed.) Data and Applications Security and Privacy XXV, Lecture Notes in Computer Science, vol. 6818, pp. 90–105. Springer Berlin (2011), http://dx.doi.org/10.1007/978-3-642-22348-8_9

[2] Barker, S., Boella, G., Gabbay, D., Genovese, V.: Reasoning about delegation and revocation schemes in answer set programming. Journal of Logic and Computation (2014)

[3] Bertino, E., Samarati, P., Jajodia, S.: An extended authorization model for relational databases. Knowledge and Data Engineering, IEEE Transactions on 9(1), 85–101 (Jan 1997)

[4] Bertino, E., Jajodia, S., Samarati, P.: A Non-timestamped Authorization Model for Data Management Systems. In: Proceedings of the 3rd ACM Conference on Computer and Communications Security. pp. 169–178. CCS '96, ACM, New York, NY, USA (1996), http://doi.acm.org/10.1145/238168.238211

[5] Chander, A., Dean, D., Mitchell, J.C.: Reconstructing trust management. Journal of Computer Security (2004)

[6] Cramer, M., Ambrossio, D.A., van Hertum, P.: A Logic of Trust for Reasoning about Delegation and Revocation. In: Proceedings of the 20th ACM Symposium on Access Control Models and Technologies. pp. 173–184 (2015), http://doi.acm.org/10.1145/2752952.2752968

[7] Fagin, R.: On an Authorization Mechanism. ACM Trans. Database Syst. 3(3), 310–319 (Sep 1978), http://doi.acm.org/10.1145/320263.320288

[8] Gomes, C.P., Kautz, H., Sabharwal, A., Selman, B.: Satisfiability Solvers. In: Van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation, pp. 89–134. Elsevier (2008)

[9] Griffiths, P.P., Wade, B.W.: An Authorization Mechanism for a Relational Database System. ACM Trans. Database Syst. 1(3), 242–255 (Sep 1976), http://doi.acm.org/10.1145/320473.320482

[10] Hagström, Å., Jajodia, S., Parisi-Presicce, F., Wijesekera, D.: Revocations – A Classification. In: Proceedings of the 14th IEEE Workshop on Computer Security Foundations. pp. 44–. CSFW '01, IEEE Computer Society, Washington, DC, USA (2001), http://dl.acm.org/citation.cfm?id=872752.873508

[11] Kleinberg, J., Tardos, É.: Algorithm Design. Pearson International Edition (2006)

[12] Li, N., Grosof, B.N., Feigenbaum, J.: Delegation Logic: A Logic-based Approach to Distributed Authorization. ACM Transaction on Information and System Security (2003)

[13] Ruan, C., Varadharajan, V.: Resolving Conflicts in Authorization Delegations. In: Batten, L.M., Seberry, J. (eds.) ACISP. Lecture Notes in Computer Science, vol. 2384, pp. 271–285. Springer (2002), http://dblp.uni-trier.de/db/conf/acisp/acisp2002.html#RuanV02

[14] Tamassia, R., Yao, D., Winsborough, W.H.: Role-Based Cascaded Delegation. In: Proceedings of the 9th ACM symposium on Access control models and technologies (2004)

[15] Yao, D., Tamassia, R.: Compact and Anonymous Role-Based Authorization Chain. ACM Transactions on Information and System Security (2009)

[16] Zhang, L., Ahn, G.J., Chu, B.T.: A rule-based framework for role-based delegation and revocation. ACM Transactions on Information and System Security (2003)