PeerAccess: A Logic for Distributed Authorization

Marianne Winslett
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
winslett@cs.uiuc.edu

Charles C. Zhang
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
cczhang@cs.uiuc.edu

Piero A. Bonatti
Università di Napoli FEDERICO II
Napoli, Italy
bonatti@na.infn.it

ABSTRACT

This paper introduces the PeerAccess framework for reasoning about authorization in open distributed systems, and shows how a parameterization of the framework can be used to reason about access to computational resources in a grid environment. The PeerAccess framework supports a declarative description of the behavior of peers that selectively push and/or pull information from certain other peers. PeerAccess local knowledge bases encode the basic knowledge of each peer (e.g., Alice's group memberships), its policies governing the release of each possible piece of information to other peers, and information that guides and limits its search process when trying to obtain particular pieces of information from other peers. PeerAccess proofs of authorization are verifiable and nonrepudiable, and their construction relies only on the local information possessed by peers and their parameterized behavior with respect to query answering, information push/pull, and information release policies (i.e., no omniscient viewpoint is required). We present the PeerAccess language and peer knowledge base structure, the associated formal semantics and proof theory, and examples of the use of PeerAccess in constructing proofs of authorization to access computational resources.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; D.4.6 [**Operating Systems**]: Security and Protection - Access Control

General Terms

Security, Languages, Theory

Keywords

P2P systems, distributed authorization, logical signature, sticky policies, release policies, proof hints

1. INTRODUCTION AND RELATED WORK

Authorization approaches for distributed systems where resources are accessed across organizational boundaries have become a topic of industrial and research interest in recent years, with reputation systems and trust negotiation emerging as two particularly interesting research directions ([3, 5, 6, 9, 11, 12, 13, 16, 17, 20, 22, 23, 24, 25], to mention just a few). In attempting to build and deploy an authorization system based on trust negotiation for a particular open system (shared access to high-performance computing resources), we found that the theory developed for authorization in open systems did not include all the features that we needed to reason about the runtime behavior of the system, or to account for all the actions that parties in the system needed to take at run time. The need was particularly acute in the area of reasoning about helpful third parties at run time, such as information brokers, credential and policy repositories, and third-party authorization services. A peer Alice may need to contact several such parties as she attempts to construct a proof that she is authorized to use a particular service, and she needs a principled way to determine who to contact, what to ask for, what kind of answers to expect, and when to give up. She needs a way to explain who she is and why she is asking for help, as her intended purpose may determine whether a third party is willing to help her, or may influence the answer that it gives her. Alice also needs a way to set limits on what can be done with the personal information that she gives out, and to determine what she is allowed to do with the information that others give to her. She also needs to be able to filter out incoming information and queries that are of no interest to her (e.g., spam and porn). She needs to be able to interact successfully with parties that push information to her, and with parties that she must query to get information.

Other researchers have examined many of the separate aspects of this problem. For example, [2] studies the problem of creating distributed proofs under an information pull paradigm, when all peers cooperate to the maximum extent possible. The authors of [18] study the problem of finding needed credentials at run time, and propose a solution based on credential typing (e.g., query the issuer to find a certain kind of credential). Other papers [1, 10] present a runtime system for constructing distributed proofs of authorization, given authoritative information on where to go to find needed credentials. In a similar way, in [3, 4, 13, 14] policies encode information about where to go to find certain needed facts.

These and the other works cited above study useful pieces of the picture, but we found that the separate pieces often did not fit together to form a solution to our real-world situation. For example, real-world peers do not exhaustively try to answer all queries they receive, as in [2]. A peer may behave quite differently depending on who is asking for help and why they are asking for help. We wanted a way to talk about sticky policies [15] (release policies that are permanently attached to the information they protect), but also wanted to be able to describe non-sticky policies within the same open system. In [1], credentials have sticky release poli-

CCS'05, November 7–11, 2005, Alexandria, Virginia, USA.

cies, and those policies are propagated to all conclusions derived using those credentials. This interesting approach will be too restrictive for many situations, and it is embedded into the syntax of the language, making it hard to change. A type-based credential discovery system, as in [18], is not flexible enough to model the evolving behavior of credential and authorization servers in computational grids, where there is often no visible relationship between the party whose signature Alice wishes to have on a fact and the party Alice must go to to obtain that signature. In [3, 4, 10, 13], information on where to obtain each credential is expressed by labeling each credential occurrence in a policy with exactly one peer. Credential location hints are affixed inside the policy and must be replicated in each rule, although the strategy for finding a credential often depends on the credential class and not on the rule where the credential is referred to—a kind of replication that may introduce errors. In general, decoupling access control policies from negotiation-related decisions such as credential fetching strategies, release policies, etc., seems a good policy engineering principle, and a step towards declarative negotiation control. Close to our work, [21] proposes a logic based formulation that supports delegatable authorizations; [7] adopts a metapolicy-based approach as in our paper. However, neither of them considers such features as sticky policies, credential discovery, or exposure issues. We know of no preexisting approach that allows one to reason about the runtime behavior of a very diverse set of peers, some of whom push information, some pull information, and some mix the two paradigms. Further, preexisting work did not consider potential interactions between various features (e.g., sticky policies on hints about where to go to obtain information about sticky policies).

In this paper, we propose the PeerAccess framework, which provides an infrastructure and language to model and reason about distributed authorization in open systems. As the framework is broad and generic, we only present an instantiation of it that is suitable for use in reasoning about access to a grid of computational resources. We introduce the architecture and describe the language in section 2, then present PeerAccess knowledge bases in section 3, release policies in section 4, semantics and fixpoint characterization in section 5, proof theory in section 6, proof hints and queries in section 7, and finally conclude in section 8.

2. FRAMEWORK AND LANGUAGE

The PeerAccess framework supports a possibly infinite set of peers, each with its own separate knowledge base (KB) of policy-related information (figure 1). Peers communicate with one another by pushing information in messages, or by pulling information through queries. The high-level behavior of each peer (i.e., what information it pushes and to whom, whose queries it tries to answer, how hard it tries to answer them, and the kinds of answers it gives) is determined by declarative event-condition-action rules for that peer. The lower-level behavior of each peer is determined by the contents of its KB, which include its own local knowledge and information that it has received from others. Its KB includes tight controls on what information it can send out or receive in messages, and hints regarding what peers to contact for help if it is trying to prove certain types of conclusions. At a high level, the language for KBs and messages can be thought of as logic programs with an open-world semantics, plus two modal operators related to the **says** operator of BAN logic [8] (to provide nonrepudiation for message contents and justification of proof results), plus a sprinkling of second-order constructs to allow declarative specification of information release policies and hints about how to construct proofs (but without introducing high runtime complexity).

The PeerAccess policy language consists of a modal language—called the *base language*—and a modal metalanguage, each with a separate countable pool of variables. Roughly speaking, the base language specifies basic access control policies and related rules; the metalanguage specifies metapolicies that determine the dynamic behavior of the system.

The *base* policy language is based on standard Datalog *atoms*, built from a countably infinite supply of constants (to model open domains). A distinguished subset of the constants, $\mathcal{N}$, contains all possible peer names. The set of predicates is application dependent. We italicize variable names, to distinguish them from constants, functions, and predicate names. At the base level, modal atoms, called *facts*, are expressions of the form "$P \operatorname{signs} \alpha$" or "$P \operatorname{lsigns} \alpha$", where $P \in \mathcal{N}$ and α is a Datalog atom. A *rule* is an expression $f_0 \leftarrow f_1 \wedge \cdots \wedge f_n$, where each f_i is a fact and $n \geq 0$. Facts are special cases of rules, where $n = 0$. If f_0 is of the form "$P \operatorname{signs} \alpha$", then the rule or fact is *directly signed* by P; otherwise, f_0 has the form "$P \operatorname{lsigns} \alpha$" and the rule or fact is *logically signed* by P. (We will omit the signatures on equality and inequality atoms, since all peers agree on the truth of such atoms.)

In the metalanguage, the set of terms includes the metavariables plus a distinct function name $\bar{s}$ for each symbol s of the two languages (variables, constants, and logical connectives), satisfying $\bar{\bar{s}} = \bar{s}$. In this way, each base or metaexpression e can be represented by a metaterm $\bar{e}$ built with the naming functions. To enhance readability, we shall simply write $\bar{e}$ as e; the context will always make clear whether e is playing the role of a term or a rule. The (nonmodal) atoms of the metalanguage are built in the usual way from metaterms and metapredicates. Facts are defined as above, i.e., as modal metalanguage atoms. Rules have the same form as above; their bodies may contain both base facts and metafacts, while the head f_0 must belong to the metalanguage. Variable instantiations must map each variable to a term of the same level, so that every instance of a well-formed expression is well-formed, too.

Each peer has a separate knowledge base (figure 1) of facts, rules, and received messages. Each KB contains the following finite sets of formulas, each described in detail in a later section:

1. Its *base policies*, which are rules over the base language.
2. All messages it ever received from other peers. Each message is a finite set of rules. (In this paper, we will not make use of the set of messages sent by a peer.)
3. Its *release policies*, containing rules about release predicates.
4. Its *hints for finding proofs*, containing rules about the 'find' metapredicate.
5. Its *exposure policies*, which act as a firewall to restrict incoming and outgoing information. To conserve space, we will not discuss exposure policies in this paper.

DEFINITION 1 (KB). *A PeerAccess global KB $\mathcal{P}$ contains one local KB for each peer:*

$$\mathcal{P} = \{(j, \mathcal{P}_j) \mid j \in \mathcal{N}\}$$

where $\mathcal{P}_j$ is peer j's local KB, i.e., the set of all messages it has received and its base, release, and proof hint policies; and $\mathcal{N}$ is the set of all peer names in the language.

We will omit "global" and "local" when referring to a KB, when the context is clear.

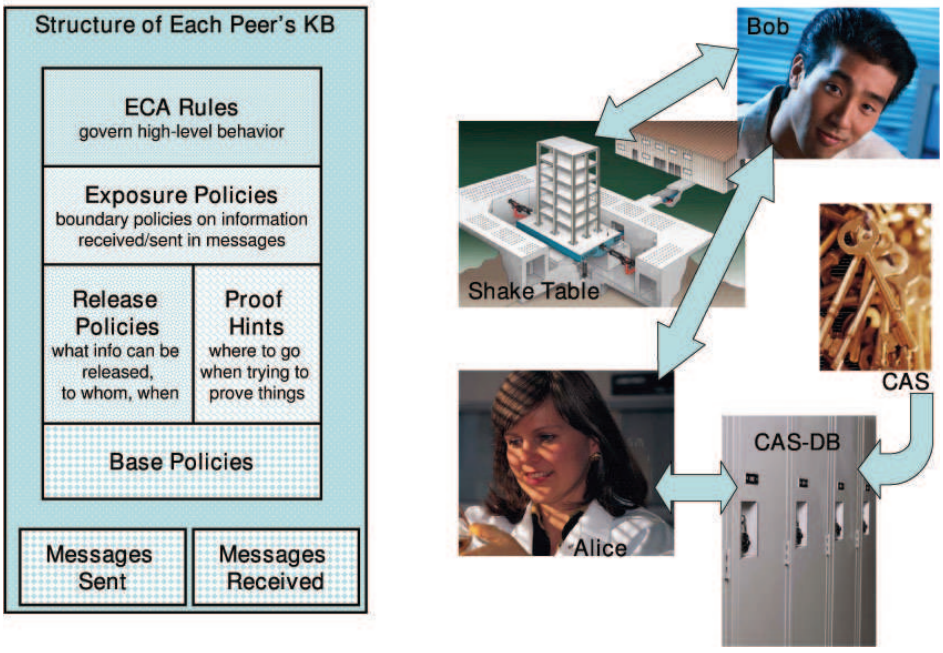

Figure 1: PeerAccess architecture and KB structure. Unidirectional arrows indicate information pushed from one peer to another. Bidirectional arrows indicate queries and responses.

3. BASE POLICIES

Intuitively, the directly signed fact "Alice signs α" in Bob's KB means that Alice has asserted in a non-repudiable manner that α holds at Alice—e.g., Alice has digitally signed α and sent out a message whose contents have eventually made their way to Bob. The logically signed fact "Alice lsigns α" in Bob's KB means that Bob has nonrepudiable evidence that leads to the conclusion that Alice would be willing to digitally sign α, if shown this evidence. The meaning of directly and logically signed rules is similar: Alice sends her directly signed rule to Bob to convince him that the logically signed counterpart of the rule (created by replacing "signs" by "lsigns" in the rule *head*) is true at Alice. We assume that whenever Alice wants to send a message to Bob, she succeeds in sending the message, Bob receives it successfully, and Bob is able to verify that its contents have not been tampered with and were actually signed by their reputed signers.

We present the formal semantics for signs and lsigns in Section 5; for the moment, it suffices to explain the three major characteristics of signs and lsigns that must hold in every KB interpretation at every peer:

1. If a directly signed rule is true at a peer Alice, then its logically signed counterpart is also true at Alice.

2. If facts f_1 through f_n and the logically signed rule $f \leftarrow f_1 \wedge \cdots \wedge fn$ are all true at a peer Alice, then f is also true at Alice.

3. If a rule logically signed by Alice is true at Alice, then so is its directly signed counterpart.

Before any peer has sent out any message, we require that each local KB contain only self-signed statements. This ensures that if Alice's KB eventually contains a fact directly signed by Bob, then Bob's KB does also.

Our running example models the behavior of the Community Authorization Service (CAS) [19] under several different possible trust assumptions. CAS is a third-party authorization service that simplifies the task of making a resource available on a high-performance computing grid, by offloading authorization reasoning from the resource manager to CAS. For example, consider the shake table, an earthquake simulation device that is managed by Bob, under the following possible scenarios.[1]

Example 1a. In this example, Bob owns and brokers all access to the shake table, and makes and directly signs his own authorization decisions. In particular, Alice will be able to access the shake table if "Bob signs auth(shaketable, Alice)" is true at Bob. Bob may store a list of authorized users/groups internally, or he may delegate his reasoning to CAS as follows:

Bob:
Bob lsigns auth(shaketable, X) ← CAS signs auth(shaketable, X)

Bob's base policy says that he will give Alice access if he has a statement directly signed by CAS, saying that Alice is authorized. If Bob's KB interpretation satisfies "Bob lsigns auth(shaketable, Alice)", then it also satisfies "Bob signs auth(shaketable, Alice)".

Example 2a. Let us change Bob's KB by one letter:

Bob:
Bob lsigns auth(shaketable, X) ← CAS lsigns auth(shaketable, X)

Now Bob wants a logical signature on CAS's proof of authorization, rather than requiring a direct signature from CAS. In other

[1]The examples will only be fully meaningful to the reader after we have presented the PeerAccess formal semantics. Conversely, the formal semantics will be very hard to follow unless the reader has an intuition about what PeerAccess is trying to accomplish. We resolve this impasse by presenting simple examples before the semantics.

words, Bob is now asking for a proof that would convince CAS that Alice is authorized to access the shake table. The pieces of the proof need not come directly from CAS. For example, for greater protection against attack, CAS could pre-sign its authorization-related rules and facts off line, and push them to a repository CAS-DB that does not have access to CAS's private keys. Then CAS-DB's base policies and received messages can be as follows:

CAS-DB:
CAS signs auth(shaketable, X) $\leftarrow$
CAS signs authgroup(shaketable, G) $\wedge$ CAS signs member(G, X)
CAS signs authgroup(shaketable, earthquake)
CAS signs member(earthquake, Alice)

To convince Bob that Alice can access the shake table, it suffices to send Bob a message containing CAS-DB's rule and facts. Once Bob receives this message and incorporates its contents into his KB, the three principles outlined earlier guarantee that CAS lsigns auth(shaketable, Alice) is true at Bob.

Example 3a. If the body of CAS-DB's rule uses 'lsigns' instead of 'signs', then we have the possibility that the proof of group membership is defined by other rules:

CAS-DB:
CAS signs auth(shaketable, X) $\leftarrow$
CAS lsigns authgroup(shaketable, G) $\wedge$ CAS lsigns member(G, X)
CAS signs member(G, X) $\leftarrow$
O lsigns member(G, X) $\wedge$ CAS lsigns owner(G, O)
CAS signs authgroup(R, G) $\leftarrow$
O lsigns authgroup(R, G) $\wedge$ CAS lsigns owner(R, O)

Here CAS is not responsible for maintaining the lists of current group members or authorized groups. Instead this task is delegated to the owners of each group and resource. The group owners may have cached their signed group membership lists at CAS-DB, or may provide them on demand to CAS-DB or to the group members, as discussed later.

4. RELEASE POLICIES

In PeerAccess, peers exchange information by sending messages to one another. Every fact and rule that a peer Alice sends out in a message must be true at Alice and must also be *releasable*. A release policy signed by peer P gives the conditions under which P thinks that it is permissible for a fact or rule ϕ to be sent in a message from peer S to peer R. In this paper, we will consider release policies over the srelease (sticky-release) predicate, which take the following form, and its logically signed counterpart:

P signs srelease $(\phi, S, R) \leftarrow f_1 \wedge \cdots \wedge f_n$,

where P, S, and R are peer names or variables; ϕ is a term over the release policy language (e.g., a base rule or a proof hint (defined later)); and f_1 through f_n are facts, with $n \geq 0$. Intuitively, srelease semantics stipulate that a peer Alice can send peer Carla a fact or rule ϕ directly signed by Bob if (1) ϕ is true at Alice, and (2) "Bob lsigns srelease(ϕ, Alice, Carla)", "Carla = Alice", or "Carla = Bob" is true at Alice. In other words, Alice can only send out a formula signed by Bob if she is sending it to herself or to Bob, or she can prove that Bob thinks that it is okay for her to send the message out. Further, Alice can only send out facts and rules that she believes to be true.

The srelease policies are *sticky*: the signer of a particular piece of information retains control over its future dissemination to other peers. (Of course, a malicious peer can choose to violate the conditions in a sticky policy, if it is not afraid of the potential legal and social ramifications of doing so.) Sticky policies are desirable and even legally mandated in many situations, but other situations may require a graceful approach to declassification of information, or even stronger control over the use of released information (e.g., control over the dissemination of conclusions reached by using that information). In the PeerAccess framework, additional release predicates can be defined to fit the needs of a particular set of peers, including limited forms of declassification and the ability to spread rumors (lsigned and unsigned formulas) and lies (formulas not true locally). For example, a direct but unsigned communication "auth(Alice, shaketable)" from CAS might convince Bob that "CAS lsigns auth(Alice, shaketable)" is true, but CAS could repudiate such a statement, and Bob would be unable to use CAS's message to convince a third party that "CAS lsigns auth(Alice, shaketable)" is true. While these variations are interesting in their own right, in this paper we confine our attention to the release of directly signed rules.

The three principles given earlier regarding the meaning of signatures on base facts and rules also apply directly to release facts and rules. We also have two additional principles to govern the release of srelease policies. In general, release policies may contain sensitive information and should not be indiscriminately released. We *do not* allow the user to define explicit KB policies governing the releasability of srelease policies, because srelease is intended to be so simple to use that peers will never have to define such policies. Instead, the releasability of srelease formulas is defined by two principles, which will be formalized later on. Oversimplifying slightly, the first principle allows Alice to send Carla an srelease rule ϕ that is directly signed by Bob if ϕ is true at Alice and ϕ's head is of the form "Bob signs srelease(α, Carla, -)", where α is directly signed by Bob. The intuition is that if Bob authorizes Carla to disseminate a piece of information, Bob should allow Carla to find out that she is authorized to disseminate the information. A second principle helps with longer dissemination chains: David can send ϕ to Alice if ϕ is true at David, ϕ's head has the form given above, and David knows that Alice can send ϕ to Carla.

When a peer Alice receives a message, she checks to see whether her exposure policies (not discussed in this paper) allow her to receive each rule in the message. She adds each receivable directly signed rule to her set of received messages S_{Alice}, and she adds the logically signed counterpart of that rule to the appropriate section of her knowledge base.

Let us revisit examples 1-3 to see the effect of release policies.

Example 1b. (Bob makes and signs his own authorization decisions for the resource he owns.) For Bob to be able to tell Alice that she is authorized, Bob can use rules of the form:

Bob:
Bob lsigns srelease(Bob signs auth(X, Y), Bob, Y)
Bob lsigns srelease(Bob signs auth(X, Y), Y, X)

Bob's first release fact says that he will release an authorization decision to the principal who is authorized by that decision. This allows Bob to tell Alice that she is authorized to access the shake table. However, this may not be enough for Alice to be able to use that authorization, e.g., if she has to present that authorization to the shake table herself. To do so, Alice must know that Bob says that it is okay for her to release his authorization decision to the shake table. Bob's second release fact accomplishes this goal. By the principles given above, Bob can send a directly signed version of his second release fact to Alice. If he sends her his authorization

decision and the release policy, her KB will contain:

Alice:
Bob signs auth(shaketable, Alice)
Bob signs srelease(Bob signs auth(X, Y), Y, X)

At this point, Alice can access the shake table by sending it "Bob signs auth(shaketable, Alice)", because that formula is satisfied in her interpretation and "Bob lsigns release(auth(shaketable, Alice), Alice, shaketable)" is also.

While Bob's proposed release rules are a good start, they are insufficient for use on the computational grids that CAS is designed for. The problem is that on those grids, Alice delegates her authority to a subjob, which in turn delegates its authority to a subjob, and so on, until eventually a subjob accesses the shake table. The rules given above only allow Alice to give her decision directly to one party, who cannot release it further. To allow Alice to release the authorization to someone who could in turn release it again, Bob can add the following rule to his release policies:

Bob lsigns srelease(Bob signs auth(X, Y), Z, W) $\leftarrow$ $Z \neq$ Bob

The principles outlined earlier imply that Bob can release this new release policy to anyone. If the policy is too generous for Bob's tastes, he could add restrictions on the recipient W to the body of the rule, e.g., W must be a member of NeesGrid, a friend of Bob, a proxy of Alice, etc. He could even require that he himself certify the property in question, e.g., Bob lsigns member(NeesGrid, W). Any such restrictions in the policy will also limit the set of peers that Bob can disclose the policy to.

One weakness of this version of Bob's release policies is that he does not allow Alice to impose her own additional controls on who is allowed to see "Bob signs auth(shaketable, Alice)". If the shake table were a sensitive resource, Alice might not want her authorization to be released to just anyone. To fix this problem, Bob can replace his third release policy by the following:

Bob:
Bob lsigns srelease(Bob signs auth(X, Y), Z, W) $\leftarrow$
$Z \neq$ Bob $\wedge$ Y lsigns condRelease(Bob signs auth(X, Y), Z, W)

Here Bob's original srelease condition, $Z \neq$ Bob, has been augmented with a second condition that says that the authorized principal (e.g., Alice) must also agree that Bob's statement can be released from peer Z to peer W. With this additional restriction, "Bob signs auth(shaketable, Alice)" can only be sent to additional peers when both Bob and Alice agree that it can be sent.

Even this new version of Bob's release policies might not satisfy Alice, who might wish to *unilaterally* impose additional release constraints of her own on the information from Bob that passes through her hands. For example, if Bob is a child and Alice is his mother, Bob might be willing to pass his own information on to anyone. To protect her family's privacy, however, Alice may wish to limit the further disclosure of Bob's information, at least in the case where that information has passed through her hands. Peers can impose such controls if we employ a release predicate with more restrictive semantics than srelease.

Let us now loosen the restriction that Bob makes his own authorization decisions, and have Bob delegate part of that task to CAS. To accomplish this, we add two additional rules to Bob's KB:

Bob:
Bob lsigns auth(shaketable, X) $\leftarrow$ CAS signs auth(shaketable, X)
Bob lsigns srelease(Bob signs auth(shaketable, X) $\leftarrow$
CAS signs auth(shaketable, X), Z, W)

Bob publicly declares that he relies on CAS for his authorization decisions. His first release policy allows him to send his rule to Alice when she wants to access the shake table. When Alice obtains releasable evidence from CAS that she is authorized to access the shake table (i.e, "CAS signs auth(shaketable, Alice)", along with "CAS signs srelease (CAS signs auth$(R, X), Y, Z)$"), she can present the fact "CAS signs auth(shaketable, Alice)" to Bob along with his delegation rule. (She can present the rule to him because he signed it. So, she can send the rule back to Bob even though he does not send her a copy of his release policy for the rule.) CAS's fact and Bob's rule together imply that Bob lsigns auth(shaketable, Alice), so Bob should be convinced that Alice can access the shake table. At this point, "Bob lsigns auth(shaketable, Alice)" is true at Bob, from which it follows that "Bob signs auth(shaketable, Alice)" is also true at Bob. Further, Bob's original release policy shows that the authorization "Bob signs auth(shaketable, Alice)" can be released to Alice. Bob can also send Alice his conditional release rule, which lets Alice do anything she likes with the authorization that he gives her.

Example 2b. In this case, CAS has delegated its authorization tasks to CAS-DB, and Bob wants to see a proof from CAS that Alice can access the shake table:

Bob:
Bob lsigns auth(shaketable, X) $\leftarrow$ CAS lsigns auth(shaketable, X)
Bob lsigns srelease(Bob signs auth(shaketable, X) $\leftarrow$
CAS lsigns auth(shaketable, X), Y, Z)
CAS-DB:
CAS signs auth(shaketable, X) $\leftarrow$
CAS signs authgroup(shaketable, G) $\wedge$ CAS signs member(G, X)
CAS signs authgroup(shaketable, earthquake)
CAS signs member(earthquake, Alice)

If we adopt the same approach to release rules as in Example 1b, then CAS-DB would have the release policy "CAS signs srelease(CAS signs auth$(R, X), Y, Z)$". This is not helpful, as CAS-DB cannot derive any facts of the form "CAS signs auth(R, X)"; CAS-DB can only derive logically signed authorizations. CAS needs to authorize CAS-DB to release all the details of the proof, so that CAS-DB can convince others that it is faithfully mirroring CAS's reasoning:

CAS-DB:
CAS signs srelease(CAS lsigns auth(shaketable, X) $\leftarrow$ CAS signs
authgroup(shaketable, G) $\wedge$ CAS signs member$(G, X), Y, Z)$
CAS signs srelease(CAS signs authgroup$(X, G), Z, W)$
CAS signs srelease(CAS signs member$(G, Y), Z, W)$

For brevity, we have set up these three release policies so that everything is publicly releasable. In practice, CAS would probably prefer to be less trusting of CAS-DB, and only authorize CAS-DB to release Alice's membership credential to Alice, while still allowing Alice to release it to anyone she chooses. Similarly, CAS might choose to limit the initial release of authgroup(shaketable, earthquake) to members of the earthquake group. CAS could also limit the initial release of its delegation rule so that CAS-DB can only give it to authorized shake table users, if desired. We will not write out these more restrictive rules here, because Example 1b has already shown how to write such policies.

To convince Bob that Alice can access the shake table, it suffices for Alice to send Bob a message containing CAS-DB's base rule and base facts, from which it follows that CAS lsigns auth(shaketable, Alice). Alice does not need to send release policies for CAS-DB's rules and facts, because Bob does not release them further. Because

Bob directly signs his own conclusions, he can send out that conclusion regardless of the sticky policies on the information he used to reach that conclusion.

Example 3b. In this example, CAS does not maintain the lists of current group members. CAS-DB has the three base rules defined earlier, a sprinkling of cached facts, and release policies from CAS authorizing public dissemination of the rules:

CAS-DB:
CAS signs auth(shaketable, X) ←
CAS lsigns authgroup(shaketable, G) ∧ CAS lsigns member(G, X)
CAS signs member(G, X) ←
O lsigns member(G, X) ∧ CAS lsigns owner(G, O)
CAS signs authgroup(R, G) ←
O lsigns authgroup(R, G) ∧ CAS lsigns owner(R, O)
CAS signs owner(earthquake, earthquakeOwner)
CAS signs owner(shaketable, Bob)
earthquakeOwner signs member(earthquake, Alice)
Bob signs authgroup(shaketable, earthquake)
CAS signs srelease(CAS signs auth(shaketable, X) ← CAS lsigns authgroup(shaketable, G) ∧ CAS lsigns member(G, X), Y, Z)
CAS signs srelease(CAS signs member(G, X) ←
O lsigns member(G, X) ∧ CAS lsigns owner(G, O), Y, Z)
CAS signs srelease(CAS signs authgroup(R, G) ←
O lsigns authgroup(R, G) ∧ CAS lsigns owner(R, O), Y, Z)

To convince Bob that she can access the shake table, Alice will need to convince him that CAS lsigns auth(shaketable, Alice). From CAS-DB's facts and rules, it follows that "CAS lsigns auth(shaketable, Alice)" is true at CAS-DB. However, CAS-DB can only send directly signed atoms and rules in messages, and "CAS signs auth(shaketable, Alice)" is not true at CAS-DB. Thus, if CAS-DB wants to be helpful, it must give Alice a set of atoms and rules from which it follows that CAS lsigns auth(shaketable, Alice). The release policies given above authorize CAS-DB to release all relevant information except "earthquakeOwner signs member(earthquake, Alice)" and "Bob signs authgroup(shaketable, earthquake)". To release these atoms, "earthquakeOwner lsigns srelease(earthquakeOwner signs member(earthquake, Alice), CAS-DB, Alice)" and "Bob lsigns srelease(Bob signs authgroup(shaketable, earthquake), CAS-DB, Alice)" must be true at CAS-DB. Further, for Alice to make use of the information CAS-DB gives her, she must be able to release it as well. This need implies that CAS-DB should have cached release policies from earthquakeOwner and Bob, because it will need these policies every time it receives a query about access, and it will receive such queries constantly. If for some reason such policies are not already cached at CAS-DB, CAS-DB can query for them automatically using the proof hints mechanism described in section 7. The same is true of the release policies that Alice will need for those same atoms.

The PeerAccess framework can be instantiated with a release predicate different from srelease if CAS should have the authority to override the wishes of the group and resource owner in releasing group membership lists and lists of authorized groups, or if CAS-DB's release of "O signs authgroup(R, G)" should require the permission of R's owner as well as O.

5. PEERACCESS SEMANTICS

DEFINITION 2 (POSSIBLE WORLD). *A possible world W is a set of logically signed ground facts that satisfies the Herbrand domain assumption, i.e., "$X = X$" $\in W$ for every ground choice of X, and "$X \neq Y$" $\in W$ for every pair of distinct choices of X and Y.*

The fact "CAS lsigns srelease(CAS signs auth(X, Y), Alice, Bob)" is not ground: variables and metavariables cannot occur anywhere in a ground formula or in a possible world.

DEFINITION 3 (INTERPRETATION). *A PeerAccess interpretation I is a set containing one local interpretation for each peer, i.e., $I = \{(j, I_j) \mid j \in \mathcal{N}\}$. Peer j's interpretation is $I_j = (\mathcal{W}_j, S_j)$, where $\mathcal{W}_j$ is a set of possible worlds and S_j is a set of rules directly signed by other peers.*

$\mathcal{W}_j$ is a *set* of possible worlds because correct local reasoning about authorization in an open system requires an open world assumption.[2]

We define the truth of a formula in I as follows, where A is the name of an arbitrary peer:

DEFINITION 4 ($\models$, MODEL).

1. *$I \models_A \phi$, for a logically signed ground rule ϕ of the form $f \leftarrow f_1 \wedge \cdots \wedge f_m$, iff for each world W in $\mathcal{W}_A$, either $f \in W$ or for some $1 \leq j \leq m$, $f_j \notin W$.*

2. *$I \models_A \phi$, for a logically signed non-ground rule ϕ, iff for every ground instance ϕ' of ϕ, $I \models_A \phi'$.*

3. *$I \models_A \phi$, for a directly signed rule ϕ, iff both of the following hold:*
   - *$\phi \in S_A$ iff ϕ is directly signed by A;*
   - *$I \models_A \phi'$, where ϕ' is the logically signed counterpart of ϕ.*

4. *$I \models_A \Phi$, for a set Φ of rules, iff for all $\phi \in \Phi$, $I \models_A \phi$. In this case, we say that I* **is a model of Φ at** *A and Φ* **is true in** *I_A.*

5. *$I \models \mathcal{P}$, for a global KB $\mathcal{P}$, iff for all peers $A \in \mathcal{N}$, $I \models_A \mathcal{P}_A$. In this case, we say that I* **is a model of** *$\mathcal{P}$ and $\mathcal{P}$* **is true in** *I.*

The preceding definition of an interpretation holds for any instantiation of the PeerAccess framework. For this paper's instantiation, each interpretation must also satisfy three requirements that are specific to the srelease predicate. In the statements of these requirements, A and B are arbitrary peers; C, D, and E are arbitrary peers or variables; ϕ is a rule directly signed by B; and f_1 through f_n are arbitrary facts, with $n \geq 0$. Note that the truth of an srelease rule does not depend on whether the formula to be released is true or false.

[2]With a closed world assumption, we could have the following scenario: A university U delegates all responsibility to its registrar R for determining who is a student: "U lsigns student(X) ← R lsigns student(X)". U does not maintain any lists of students itself. Under a closed world assumption, "U lsigns student(Alice) ← U lsigns student(Bob)" is true at U, because U does not know that Bob is a student. If U signs and sends this true rule to Alice, and Alice obtains proof that Bob is a student, then Alice will have a proof of U lsigns student(Alice), which U never intended. An open world assumption prevents this rule from ever being true at U (because U has possible worlds where Bob is a student), thereby preventing its dissemination in messages.

1. (A peer can send any directly signed rule to itself or to the rule's signer.) If $I \models_A \phi$, where ϕ is directly signed by B, then $I \models_A B$ lsigns srelease(ϕ, A, B) and $I \models_A B$ lsigns srelease(ϕ, A, A). The former condition guarantees that B cannot repudiate a proof provided by A, by claiming that he does not have ϕ. The latter condition avoids certain awkward theoretical situations.

2. (If Bob authorizes Carla to disseminate a piece of information under certain conditions, Bob should allow Carla to find out that she is authorized to do so.) Recall that we prohibited the user from writing srelease rules for srelease rules, because they are hard to write and understand; our intent is that such rules should be generated automatically, using this principle and the one that follows. If $I \models_A \phi$, where ϕ is of the form "B lsigns srelease(ψ, C, D) $\leftarrow f_1 \wedge \cdots \wedge f_n$", then $I \models_A B$ lsigns srelease(ϕ, A, C). Here Alice is allowed to release Bob's release rule ϕ to Carla, because ϕ authorizes Carla to release information under certain conditions.

3. (If Bob authorizes Carla to release a piece of information to Doug, and Bob also authorizes Doug to release it to Edward, then Carla is allowed to know that Doug can release this info to Edward, and the release policy itself can be forwarded from Carla to Doug.) If $I \models_A B$ lsigns srelease(ϕ, C, D) $\leftarrow f_1 \wedge \cdots \wedge f_n$ and $I \models_A \psi$, where ψ is of the form "B lsigns srelease(ϕ, D, E) $\leftarrow f_1 \wedge \cdots \wedge f_n$", then $I \models_A B$ lsigns srelease(ψ, A, C).

One can view the evolution of a global KB over time as being represented by a sequence $I^1, I^2, \ldots$, where each evolution step corresponds to a set of messages being sent in parallel between peers. Each such message can add formulas to the local KBs, but the contents of the local KBs are bounded above: there are only so many new messages that can be sent. As a result, for each legal initial global KB there is exactly one *canonical model*, written $\overline{\mathcal{P}}$, which is the interpretation representing the maximal attainable state of knowledge across all peers. (In this paper, we do not allow peers to delete formulas from their KBs, or to insert formulas other than those that arrive in messages.) Each KB also has one unique *isolated model*, written $\underline{\mathcal{P}}$, which represents each peers' maximal local knowledge before receiving any message from others. The following definitions present fundamental operations on interpretations, along with theorems and proofs regarding their properties.

DEFINITION 5 (UNION OPERATOR $\oplus$). *For two interpretations I and I' and peer A, we define*

$$I_A \oplus I'_A \;=\; ((\mathcal{W}_A \cup \mathcal{W}'_A)\;,\; (S_A \cap S'_A))$$

$$I \oplus I' = \{(B, I_B \oplus I'_B) \mid B \in \mathcal{N}\}.$$

PROPOSITION 1 (MODEL CLOSURE UNDER UNION). *The union $I \oplus I'$ of any two models of $\mathcal{P}$ is still a model of $\mathcal{P}$.*

Proof. Let I and I' be two of $\mathcal{P}$'s models, and ϕ be any rule in $\mathcal{P}_A$. It immediately follows that ϕ is true in both I and I'. Consider the following cases.

1. If ϕ is a ground logically signed rule, then ϕ is satisfied in every possible world in W_A and in W'_A; thus ϕ is satisfied in every possible world in $W_A \cup W'_A$; therefore ϕ is true in $I \oplus I'$ at A.

2. Consequently, if ϕ is a logically signed rule, all its ground instances will be true in $I \oplus I'$ at A; thus ϕ is true in $I \oplus I'$ at A.

3. If ϕ is a directly signed rule, its logically signed version is true in $I \oplus I'$ at A; if ϕ is not self-signed, then it is also in both S_A and S'_A, which means $\phi \in S_A \cap S'_A$; therefore ϕ is true in $I \oplus I'$ at A.

We conclude that $I \oplus I'$ is a model for $\mathcal{P}$.

DEFINITION 6 (INTERSECTION OPERATOR $\odot$). *For two interpretations I and I' and peer A, we define*

$$I_A \odot I'_A \;=\; ((\mathcal{W}_A \cap \mathcal{W}'_A)\;,\; (S_A \cup S'_A))$$

and further define

$$I \odot I' = \{(B, I_B \odot I'_B) \mid B \in \mathcal{N}\}.$$

PROPOSITION 2 (MODEL CLOSURE UNDER INTERSECTION). *The intersection $I \odot I'$ of any two models of $\mathcal{P}$ is still a model of $\mathcal{P}$.*

This proposition can be proved using the arguments in the proof of the previous proposition.

DEFINITION 7 ($\preceq$ RELATION). *For two interpretations I and I' and an arbitrary peer A, we define*

$$I_A \preceq I'_A \quad \textit{iff} \quad (\mathcal{W}_A \supseteq \mathcal{W}'_A) \textit{ and} (S_A \subset S'_A).$$

Then we define $I \preceq I'$ iff for all $B \in \mathcal{N}, I_B \preceq I'_B$. We call I' an **upper bound** *for I.*

The relation $\preceq$ is reflexive, transitive, and anti-symmetric, hence it is a partial order on interpretations. As examples, we have $(I \oplus I') \preceq I$, and $I \preceq (I \odot I')$. The partial order has a maximal element I^∞, in which each peer has no possible worlds and a set containing all directly signed non-self-signed rules. We further define $I \prec I'$ iff $I \preceq I'$ and $I \neq I'$.

DEFINITION 8 (ISOLATED MODEL). *The union of all models of KB $\mathcal{P}$ is its isolated model, written $\underline{\mathcal{P}}$.*

The isolated model reflects the viewpoint of each peer, considering only that peer's local knowledge. A peer's local reasoning is performed with respect to its portion of an isolated model.

DEFINITION 9 (RELEASABILITY). *A rule ϕ directly signed by peer B is releasable from peer A to C in interpretation I iff ϕ is true at I_A and $I \models_A B$ lsigns srelease(ϕ, A, C).*

DEFINITION 10 (STABILIZED INTERPRETATION). *An interpretation I is stabilized iff for all peers A, B, and C and all rules ϕ that are directly signed by B, if ϕ is releasable from A to C, then ϕ is true in I at C.*

Intuitively speaking, every interesting message has already been sent in a stabilized interpretation.

DEFINITION 11 (MESSAGES). *We define* global, local, *(*maximum*)* legal, *and* new *messages as follows:*

1. *A global message $M = \{(A, M_A) \mid A \in \mathcal{N}\}$, where M_A is a finite set of directly signed rules. A rule m in M_A is a local message for peer A. We omit the terms "global" and "local" when the intent is clear from the context.*

2. *M is a new message for interpretation I iff there exists a peer A and rule $\phi \in M_A$, such that $I \not\models_A \phi$.*

3. *M is a legal message for interpretation I iff for all peers A and all rules $\phi \in M_A$, there exists a peer B such that ϕ is releasable in I from B to A.*

4. *M is the maximum legal message for I iff for every legal message M' for I and for all peers A, $M'_A \subseteq M_A$.*

A global message is also a PeerAccess KB.

DEFINITION 12 (TRANSITION AND SUCCESSORS). *We define immediate successor relation $\Rightarrow$ (leads to), transition sequence, fairness and eagerness as follows.*

1. *Interpretation I' is an immediate successor of I, denoted $I \Rightarrow I'$, iff there exists a legal message M for I such that $(I \odot \underline{M}) = I'$. In this case, we say I* **leads** *to I' by message M, and M sends m to I' at A when $m \in M_A$. I' is a non-trivial successor of I if M is a new message for I.*

2. *A sequence $I_1, I_2, \ldots$ of interpretations is a transition sequence iff $I^1 \Rightarrow I^2 \Rightarrow \cdots$.*

3. *A transition sequence is* **fair** *iff for each choice of $j > 0$, each legal message M^j for I^j, every peer A, and every rule $m \in M^j_A$, there exists an interpretation I^l in the sequence that leads to I^{l+1} by sending legal message M^l, and $m \in M^l_A$. The intuition is that every possible local message gets sent during the sequence, within a finite number of transitions.*

An interpretation can have more than one immediate successor. Note that if $I \Rightarrow I'$, then $I \preceq I'$.

DEFINITION 13 (UPPER BOUND). *Let Seq be an infinite sequence of interpretations $I^1, I^2, \ldots$ such that $I^1 \preceq I^2 \preceq \ldots$. I is an upper bound of Seq iff for all j, $I^j \preceq I$.*

PROPOSITION 3 (UPPER BOUND UNION CLOSURE WITH $\preceq$).
If $I \preceq I^a$ and $I \preceq I^b$, then $I \preceq (I^a \oplus I^b)$.

2. *The union $L \odot L'$ of two upper bounds for an interpretation sequence Seq is still an upper bound for Seq.*

Proof. (Part 1) Let $\mathcal{W}$, $\mathcal{W}^a$, and $\mathcal{W}^b$ be the corresponding sets of possible worlds and let S, S^a, and S^b be the corresponding sets of directly signed formulas. For all choices of peers A, $I \preceq I^a$ and $I \preceq I^b$ imply that $W \supseteq W^a$ and $W \supseteq W^b$; thus $W \supseteq (W^a \cup W^b)$. We obtain $S \subseteq (S^a \cap S^b)$ similarly, and conclude that $I \preceq (I^a \oplus I^b)$. (Part 2) The desired result follows by applying Part 1 to all interpretations in the sequence.

DEFINITION 14 (FIXPOINT). *Let Seq be an infinite sequence of interpretations $I^1, I^2, \ldots$ such that $I^1 \preceq I^2 \preceq \ldots$. The union I^* of all upper bounds for Seq is Seq's fixpoint.*

For every upper bound I for Seq, $I^* \preceq I$. Note that Seq does not necessarily include the fixpoint.

Let Seq be an infinite transition sequence $\underline{\mathcal{P}}, I^1, I^2, \ldots$. We say that Seq is a transition sequence for $\mathcal{P}$.

DEFINITION 15 (CANONICAL MODEL). *$\mathcal{P}$'s canonical model, written $\overline{\mathcal{P}}$, is the intersection of the fixpoints of all the transition sequences for $\mathcal{P}$.*

THEOREM 1 (PEERACCESS CONFLUENCE THEOREM). *For each infinite transition sequence Seq for $\mathcal{P}$ and its fixpoint I^*, we have:*

1. *$I^* \preceq \overline{\mathcal{P}}$.*

2. *$I^* = \overline{\mathcal{P}}$ iff Seq is a fair transition sequence.*

Proof. (Part 1.) By definition, $\overline{\mathcal{P}}$ is the intersection of I^* and others, thus $I^* \preceq \overline{\mathcal{P}}$.

(Part 2.) Represent any two of these fair sequences E and F as $E^1 = \overline{\mathcal{P}}, E^2, \ldots$, and $F^1 = \overline{\mathcal{P}}, F^2, \ldots$.

Let m be a member of the set of directly signed formulas in sequence E's fixpoint E^* at peer A. Then m has to be in some E^i's set of directly signed formulas at A. Otherwise, we can have an upper bound E^+ that has the same possible worlds and set of directly signed formulas as E^*, except that m is not in E^+'s set of directly signed formulas at A. In that case $E^+ \succ E^*$, which contradicts the fact that E^* is the fixpoint. So we can assume m first appears in E at E^i and is sent from B at E^{i-1}.

Let the tuple (x, P, A), called a *message delivery*, represent the fact that that peer P sent a local message x to A at transition E^{i-1}. Let S be an empty stack, and push (m, B, A) onto S. Then for each peer P that sent B a message during stage E^{i-2}, push a tuple (x, P, B) onto the stack. Repeat the process for each message that a peer P' sent to P or B at stage E^{i-3}. Continue the process back through each stage, pushing messages sent by (potentially more and more) peers, until all relevant messages from the first stage have been pushed.

The resulting stack shows how m came to be delivered to A. Each message in the stack is legal (releasable) if all messages above it in the stack have already been sent. If we pop messages off the stack one by one, we get a finite sequence of legal message deliveries, (m^1, B^1, A^1), (m^2, B^2, A^2), ..., (m, B, A.) Because $F1$ equals $E1$, m^1 must be releasable at F^1 from A^1 to B^1. Under the fairness assumption, this legal message must be sent at some point in F; say it happens at F^r. After that point, m^2 is releasable from A^2 to B^2, and the same argument as used for m^1 implies that m^2 is eventually sent in F. Repeating this argument, we find that eventually m is sent in F; say this happens in F^k at A. Then m is also in F^* at A, as otherwise we would have a fixpoint $F^+ \succ F^*$. We conclude that E^* and F^* have the same sets of directly signed formulas.

E and F start with the same set of possible worlds at A, and every time a message is received at A, the elimination of possible worlds (by intersecting with all possible worlds implied by the received messages at A) is determined by the set of messages in A. If a possible world w is in E^1 at A, but not in E^* at A, then it must be eliminated from some E^i at A because of a message m received by E^i at A. As we have proved, m also gets delivered to some F^j, which eliminates w from F^j at A and all F^j's successors. Thus w is not in F^* at A either. We conclude that $E^* = F^*$, which means that all of $\mathcal{P}$'s fair transition sequences have the same fixpoint, written $\mathcal{F}$.

We can use the same arguments to show that every local message sent to A in an unfair transition sequence is also present in the set of directly signed formulas at A in the fixpoint of a fair transition sequence, but not the other way around. Thus any unfair transition sequence's fixpoint $\mathcal{F}' \prec \mathcal{F}$, which means $\mathcal{F} \odot \mathcal{F}' = \mathcal{F}$. As $\overline{\mathcal{P}}$ is defined as the union of all transition sequences' fixpoints, we conclude that $\overline{\mathcal{P}} = \mathcal{F}$.

For the other direction, we have already proved that for any unfair sequence for $\mathcal{P}$ and resulting fixpoint $\mathcal{F}'$, we have $\mathcal{F}' \succ \overline{\mathcal{P}}$. It follows that if a sequence's fixpoint is $\overline{\mathcal{P}}$, then this sequence must be a fair sequence.

6. PROOF THEORY

We now turn our attention from what is *true* in the interpretations of PeerAccess knowledge bases to what is *provable.*

DEFINITION 16 ((LOCAL) DERIVATION). *We have the following* local *derivation rules to derive new information inside a particular peer A's KB, $\mathcal{P}_A$:*

- **Instantiation.** *From a logically signed rule ϕ in $\mathcal{P}_A$, derive an instance of ϕ.*
- **Modus ponens.** *From the logically signed rule $f \leftarrow f_1 \wedge \cdots \wedge f_m$ and facts f_1 through f_m, derive f.*
- **Signature.** *From a rule logically signed by A, derive its directly signed counterpart.*
- **Self-release.** *From a rule ϕ directly signed by B, derive "B lsigns srelease(ϕ, A, B)" and "B lsigns srelease(ϕ, A, A)".*
- **srelease-1.** *From a rule ϕ of the form "B lsigns srelease(ψ, C, D) $\leftarrow f_1 \wedge \cdots \wedge f_n$", derive "$B$ lsigns srelease(ϕ, A, C)", where B, C, and D are arbitrary peers.*
- **srelease-2.** *From a rule of the form "B lsigns srelease(ϕ, C, D) $\leftarrow f_1 \wedge \cdots \wedge f_n$" and a rule ψ of the form "B lsigns srelease(ϕ, D, E) $\leftarrow f_1 \wedge \cdots \wedge f_n$", derive "$B$ lsigns srelease(ψ, A, C)", where B, C, D, and E are arbitrary peers.*

A sequence $\mathcal{P}_A^1, \ldots, \mathcal{P}_A^n$ of peer A's KBs is a (local) derivation sequence *for peer A if $\mathcal{P}_A^{i+1}$ can be obtained from $\mathcal{P}_A^i$ by applying at most one derivation rule, for all $1 \leq i < n$.*

DEFINITION 17 ((GLOBAL) DERIVATION). *For peers A and B with KBs $\mathcal{P}_A$ and $\mathcal{P}_B$ respectively, we have the following global derivation rule.*

- **Message.** *From a set $\Phi \subseteq \mathcal{P}_A$ of rules such that for each $\phi \in \Phi$, ϕ is directly signed by some peer C and "C lsigns srelease(ϕ, A, B)" $\in \mathcal{P}_A$, derive Φ in $\mathcal{P}_B$.*

A sequence $\mathcal{P}^1, \ldots, \mathcal{P}^n$ of global KBs is a (global) derivation sequence *if for all peers A and all $1 \leq i < n$, either (1) the sequence $\mathcal{P}_A^i, \mathcal{P}_A^{i+1}$ is a local derivation sequence for A, or (2) $\mathcal{P}_A^{i+1}$ can be obtained from $\mathcal{P}_B^i$, for some peer B, through an application of the message derivation rule.*

DEFINITION 18. *A KB $\mathcal{P}$ derives a rule ϕ at peer A, written $\mathcal{P} \vdash_A \phi$, iff there exists a derivation sequence $\mathcal{P}^1, \ldots, \mathcal{P}^n$ such that $\phi \in \mathcal{P}_A^n$. Then $\mathcal{P}^1, \ldots, \mathcal{P}^n$ is a* proof *for ϕ at A. When only local derivation rules are used in this proof, we write $\mathcal{P}_A \vdash_A \phi$.*

Proofs that use only local derivations describe the access control process from a particular peer's point of view. Proofs that use global derivations describe what could happen in the system, so are useful for analyzing safety and liveness. We next argue that each KB's local proofs are *sound* and *complete* for the isolated model $\overline{\mathcal{P}}$; and accordingly, so are the global proofs for the canonical model $\overline{\mathcal{P}}$.

THEOREM 2 (LOCAL SOUNDNESS). *For any KB $\mathcal{P}$, rule ϕ, and peer A, if $\mathcal{P}_A \vdash_A \phi$, then $\overline{\mathcal{P}} \models_A \phi$.*

Proof. When only the local derivation rules are used, derivation in PeerAccess becomes similar to derivation in an ordinary logic program. We prove its local soundness by arguing for the soundness of each derivation rule.

- **Instantiation.** If a logically signed rule ϕ is true in $\overline{\mathcal{P}}$ at A, then every instance of it is present in every possible world of $\overline{\mathcal{P}}$ at A, so every instance of it is also true in $\overline{\mathcal{P}}$ at A.
- **Modus ponens.** For every logically signed rule ϕ of the form $f \leftarrow f_1 \wedge \cdots \wedge f_m$ and facts f_1 through f_m, let f_1' through f_m' be corresponding ground instances. For every world w of $\overline{\mathcal{P}}$ at A, as all f_1' through f_m' are present in w, if f' is not in w, then the rule ϕ is not true in w, which is a contradiction. We conclude that f' is in w and f is true in $\overline{\mathcal{P}}$ at A.
- **Signature.** If a ground rule ϕ logically signed by A is true in $\overline{\mathcal{P}}$ at A, the model theoretic definition of directly signed rules tells us that ϕ's directly signed counterpart is also true at A.
- **Release.** The remaining local derivation rules are for the srelease predicate, and their soundness follows immediately from the model theoretic constraints on srelease.

Thus ϕ is also true in $\overline{\mathcal{P}}$ at A, and we conclude that $\overline{\mathcal{P}} \models_A \phi$.

THEOREM 3 (LOCAL COMPLETENESS). *For any KB $\mathcal{P}$ and ground rule ϕ, if $\overline{\mathcal{P}} \models_A \phi$, then $\mathcal{P}_A \vdash_A \phi$.*

Proof. Since peer A has a finite set of local rules at each point, this completeness result can be shown in the same way as the completeness results for general logic programs, with the exception of proofs regarding the srelease predicate. For that predicate, each point of its model-theoretic definitions corresponds directly to a proof-theoretic counterpart, so the completeness of reasoning about srelease follows immediately from the definition of $\vdash$.

THEOREM 4 (GLOBAL SOUNDNESS). *For any KB $\mathcal{P}$, peer A, and rule ϕ, if $\mathcal{P} \vdash_{\mathcal{A}} \phi$, then $\overline{\mathcal{P}} \models_A \phi$.*

Proof. We prove this theorem by induction on the number of steps of the proof. If the proof has one step, then $\phi \in \mathcal{P}$, and the theorem holds. Otherwise, assume that it holds for all proofs of length less than n, and now consider the nth and final step in deriving ϕ at peer A.

1. When a local derivation rule is used to derive ϕ, the Local Soundness Theorem tells us that ϕ is true in $\overline{\mathcal{P}}$ at A. As $\overline{\mathcal{P}} \preceq \overline{\mathcal{P}}$, we know ϕ is true in $\overline{\mathcal{P}}$ at A.
2. When a global derivation rule is used to derive ϕ, peer A gets a message m containing ϕ. In this case, $\mathcal{P}$ becomes $\mathcal{P}'$ after the message is received. Let M be a global message with $M_A = \{m\}$, and with the empty set for all other local messages. Then $\overline{\mathcal{P}'}$ becomes $\mathcal{P} \odot M$ after M is received, and m is true in $\overline{\mathcal{P}'}$ at A. By the Fixpoint Theorem, we know that $\overline{\mathcal{P}'} \preceq \overline{\mathcal{P}}$, so m is also true in $\overline{\mathcal{P}}$ at A.

By the induction hypothesis, we conclude that the theorem holds for proofs of all lengths.

THEOREM 5 (GLOBAL COMPLETENESS). *For any KB $\mathcal{P}$ and ground rule ϕ, if $\overline{\mathcal{P}} \models_A \phi$, then $\mathcal{P} \vdash \phi$.*

Proof. Suppose that ϕ is true in the canonical model $\overline{\mathcal{P}}$. Consider a fair transition sequence $Seq = I^1, I^2, \ldots$ with fixpoint $\overline{\mathcal{P}}$; such a sequence must exist, because ϕ is ground and $\mathcal{P}$ is finite. Let j be the first point in Seq in which ϕ is true in I_A^j. If $j = 1$, then we have ϕ in the initial local interpretation of $\mathcal{P}_A$, and by the Local Completeness Theorem, the current theorem follows. Otherwise, by the induction hypothesis, let us assume that for every rule ψ

that is true at some peer B in I^k, for $1 \leq k < j$, the theorem holds; in other words, we have a proof of ψ at B in the canonical model. By the definition of a transition sequence, a finite set of rules must have been sent to peer A in a new legal message, causing the transition from I^{j-1} to I^j. Consider any member r of this set that was not already in the set of directly signed non-self-signed rules in I^{j-1}. Recall that B can only send r to A if r is directly signed and releasable, i.e., r is true at B and either A signed r, $A = B$, or "C lsigns srelease(r, B, A)" is true at B. In the current situation, $A \neq B$, because r is not in the set of directly signed non-self-signed rules in I^{j-1}. Similarly, A cannot be the signer of r. Thus it must be the case that "C lsigns srelease(r, B, A)" is true at B in I^{j-1}. By the induction hypothesis, we have a proof for r at B in the canonical model of $\mathcal{P}$. By the induction hypothesis, we also have a proof in the canonical model for every rule r that was true at A in I^{j-1}.

Consider the set m of all new directly signed non-self-signed rules that arrived at A during the transition between I^{j-1} and I^j. Given that ϕ is true in I^j but not in I^{j-1}, one possibility is that ϕ is in m. In that case, by the induction hypothesis, we have a proof of ϕ at B in the canonical model. By the message derivation rule, we also have a proof of ϕ at A in the canonical model.

Otherwise, we have $\phi \notin m$, so ϕ must not be directly signed by another peer. Consider the KB K that consists of the initial KB $\mathcal{P}$, plus every legal new message sent in the transitions up to and including the transition to I^j. The Local Completeness Theorem tells us that if ϕ is true at A in the isolated model of K, then there is a proof of ϕ at A for K. ϕ must be true in the isolated model of K, because K captures all the messages sent to any peer since the first transition of the system. Further, by the induction hypothesis, we have a proof of every formula that is in K but is not in $\mathcal{P}$. We can take the proof of ϕ in K, and extend it by prefacing it with the proofs of all the formulas in K that are used in the proof of ϕ in the isolated model of ϕ but are not present in $\mathcal{P}$ (by the definition of a proof, this set of formulas must be finite), to create a proof of ϕ in the canonical model of $\mathcal{P}$.

7. HINTS FOR PROOFS AND QUERIES

The preceding sections have talked about what is true and provable at peers, without considering whether the peers are willing to construct the proofs or determine the truths. If Alice needs to determine whether ϕ is true and is unable to do so on her own, in some applications Alice could ask every other peer in the system for help. However, in the real world there are typically so many peers that Alice would not want to take the time to ask all of them for help, and most peers would be unwilling to help her anyway. Alice uses her *proof hints* to restrict her search to peers where she has a reasonable chance of getting help. For this purpose, each PeerAccess peer's knowledge base contains a section devoted to proof hints, which are metalevel facts and rules that suggest which peers Alice should ask for help as she tries to determine whether certain atoms and rules are true.

Each proof hint takes the form "A signs find(ϕ, B, C) $\leftarrow f_1 \wedge \cdots \wedge f_n$," or its logically signed counterpart, where A is a peer name, B and C are peer names or variables, "find" is a metalevel proof hint predicate, ϕ is a rule, and f_1 through f_n are base or proof hint facts. Intuitively, if a peer Alice is trying to prove ϕ, the hint "Bob signs find(ϕ, Alice, Carla) $\leftarrow f_1 \wedge \cdots \wedge f_n$" means that Bob suggests that Alice ask Carla about ϕ, under conditions f_1 through f_n. Much as we disallowed srelease policies for srelease policies, we disallow proof hints for finding proof hints, because the additional layer of indirection adds no interesting expressive capability and complicates execution at run time.

Alice can define broker predicates and use them to describe how to use them to find proof hints:

Alice:
Alice lsigns NeesGridBrkr(David)
Alice lsigns NeesGridBrkr(Edith)
Alice lsigns find(O lsigns auth(R, X), Alice, B) $\leftarrow$
Alice lsigns NeesGridResource(R) $\wedge$ Alice lsigns NeesGridBrkr(B)

These formulas say that when Alice is trying to prove that she or her proxy is authorized to access a NeesGrid resource, she should ask one of the NeesGrid brokers for help, by sending the broker her query. If Alice asks David whether Alice can access the shake table (using the query ?O lsigns auth(shaketable, Alice), as defined in the next section), David could in theory respond with a yes or no answer. However, if David really is a broker, he will not answer the query directly. Instead David will give Alice a new proof hint, e.g., "David signs find(Bob signs auth(shaketable, Alice), Alice, Bob)".

As another example, for CAS-DB to ask resource owners for permission to release atoms about their authorized groups, CAS-DB could use the following proof hint:

CAS-DB lsigns find(
O signs srelease(O signs authgroup(R, G), CAS-DB, X),
CAS-DB, O)
$\leftarrow$ CAS lsigns owner(R, O)

If David sends Alice a proof hint, then the proof hint must have been releasable. The releasability of proof hints is determined exactly as for base predicates, and the semantics and derivation rules presented in the previous sections remain unchanged; the truth of a proof hint formula does not depend on whether the formula to be proved is true or false. The impact of proof hints lies in their effect on a peer's run-time behavior, which is a tunable feature of the PeerAccess framework. For example, proof hints can be used to encode a more modular version of the credential discovery type system of [18], as rules that say that if a credential is of type issuer-traces-all, we should ask by asking the prospective signer for the credential; if it is of type subject-traces-all, we should start by asking the prospective owner for it; and so on. Whether and how a peer makes use of proof hints is governed by the event-condition-action rules for that peer in the PeerAccess framework. For the purposes of this paper, we will use a single such rule for all peers: when Alice is unable to make headway on determining the truth of a fact, she does not give up until she has asked for help by querying each peer recommended by any proof hint in her local knowledge base. More precisely, if Alice is trying to determine whether ϕ is true and is unable to do so using her local knowledge base, she will send the query ?ϕ to each peer P such that P' lsigns find(ϕ, Alice, P) is true at Alice, for any peer P'. In this paper, Alice will not ask other peers for additional proof hints for ϕ if the hints that she has do not lead to proofs of ϕ.

In even the smallest examples, proof hints of the form "Alice lsigns find(P lsigns Φ, Alice, P)", where Φ is a metavariable, would cause Alice to issue a huge number of queries whenever she got stuck during proof construction—queries not only about each leaf of the proof tree under construction, but also about each interior node. To protect Alice from a denial of service attack by purveyors of proof hints that instruct Alice to ask all peers, Alice should use her exposure policies to limit the set of proof hints she allows into her KB, and she should use ECA rules that require the signer of a proof hint that she acts upon to have a good rating from a reputation service that she trusts (modeled as an additional peer or peers who sign ratings). Even in the small examples used in this

paper, we cannot allow Bob to have a proof hint that directs him to always ask the signer of a fact for help when trying to prove the fact—this would cause him to contact CAS or CAS-DB himself, rather than having Alice do the work for him. We will show how to use proof hints in the CAS examples once we have discussed the format and handling of queries.

In PeerAccess, peer ECA rules control the choice of "pull" or "push" paradigms of information dispersal. For the pull paradigm, a query takes the form $?f_1 \wedge \cdots \wedge f_n$, where each f_i is a fact or a rule delimited by parentheses, for $1 \leq i \leq n$. The meaning of a ground conjunctive formula is defined in the traditional manner: the formula is true at a peer if all conjuncts are true at that peer. In this paper, we will assume that all queries are releasable, i.e., A lsigns srelease(?Φ, X, Y) is true at every peer A, and we will omit the definition of query releasability.

The conjunctive form of a query allows Alice to ask CAS-DB whether CAS signs a particular fact, and include a statement about the purpose that she intends to use that signed fact for (in the form of a proposed release policy for the fact). For example, Alice may query CAS-DB with ?(CAS lsigns auth(shaketable, Alice) $\wedge$ (CAS lsigns srelease(CAS lsigns auth(shaketable, Alice), Alice, shaketable))). A more voluminous set of rules in the query would allow Alice to explain that she will only give CAS's authorization statement to her proxies and to the shaketable. Because PeerAccess peers can choose to ignore queries, a peer may choose not to respond to a query that lacks an acceptable purpose. If it is important to support nonrepudiation of queries (e.g., for legal purposes), then we can require that queries be signed; in this paper, we do not consider that option.

The run-time behavior of a set of peers, as encoded in their proof hints, exposure policies, and ECA rules, depends on the peers' designers' choice of run-time strategies, such as the proposals put forth by [2, 4, 14, 25]. Different strategies have different conventions for what the acceptable responses are to a query. For example, SD3 adopts the convention that Bob's response must be such that Alice never has to ask Bob the same query again as she continues to work on getting all the answers to her query [14]. The proposal of [2] guarantees complete query answers, under an assumption that peers are fully cooperative. We intend PeerAccess to be customizable to support all of these proposed strategies and the many others that will be proposed in the future; each such proposal can guarantee (or not) properties such as termination, safety, and liveness in its own way. Thus the only query answer requirement PeerAccess imposes is that every answer must be an ordinary message (directly signed, releasable, and true at the sender). This allows Bob's query-answering behavior to range from non-response to sending back all releasable information already in his KB plus *everything* he can glean from other peers, whether or not it is relevant to the query. In our remaining space, we cannot investigate any strategy in detail, but we will revisit example 1 to see the effect of proof hints and queries on an SD3-like run-time strategy.

Example 1c. (Bob makes and signs his own authorization decisions, relying on directly signed CAS statements in his internal reasoning.) Alice starts the interaction by sending Bob the query ?Bob lsigns auth(shaketable, Alice). Bob's KB contains the following, plus three additional release rules for the auth predicate:

Bob:
Bob lsigns auth(shaketable, X) $\leftarrow$ CAS signs auth(shaketable, X)
Bob lsigns find(CAS signs auth(shaketable, X), X, CAS)
$\leftarrow X \neq$ Bob
Bob lsigns srelease(
Bob signs find(CAS signs auth(shaketable, X), X, CAS)
$\leftarrow X \neq$ Bob, Y, Z)

Bob's exposure policies allow him to receive queries about shake table authorizations from individual parties who would like to be authorized. Bob is configured so that he tries to prove "Bob signs auth(shaketable, Alice)" when he receives Alice's query.

Bob checks to see if "Bob lsigns auth(shaketable, Alice)" is already in his KB (signature derivation rule), and finds that it is not. Next he looks for rules that will allow him to expand the lsigned version of his goal (modus ponens derivation rule), and finds his CAS delegation rule. Then his effort shifts to proving "CAS signs auth(shaketable, Alice)", which is not in his KB. It is not a self-signed formula, so an lsigned version of the formula would not help. He has no rules that allow him to expand this proof goal. Bob is stuck, and there are no other rules that allow him to expand his original proof goal.

Since his local proof attempts have failed, Bob looks for proof hints in his KB that will tell him how to prove any of his proof goals, or that suggest sources for new rules to use in expanding his current set of proof goals. He has only one proof hint, and its preconditions are not satisfied. Bob is not configured to look for additional proof hints at run time, so his proof attempts have ended in failure. This is exactly the desired outcome: Bob wants Alice to do the work of querying CAS. In accordance with SD3's principles, Bob sends Alice sufficient information that she will not have to ask him the same query again (except to get his direct signature on his authorization); he sends her "Bob signs auth(shaketable, X) $\leftarrow$ CAS signs auth(shaketable, X)", after proving that this formula is releasable (signature rule). Bob is configured to send along all releasable proof hints that are possibly relevant to his answers, so he also sends his proof hint. (It would not be unreasonable in this case for Bob to be configured to send Alice every releasable formula in his KB. Or Bob might respond with the counterquery ?CAS signs auth(Alice, shaketable).)

Alice is configured with an exposure policy that allows her to accept Bob's query and his associated proof hint, which she adds to her KB. In attempting to answer Bob's query, her local knowledge immediately fails her and she makes use of Bob's proof hint, which tells her to query CAS. CAS accepts queries from parties who are asking whether they are authorized to access resources that CAS knows about. Thus CAS accepts Alice's query, and tries to prove "CAS signs auth(shaketable, Alice)" using local inference. If CAS answers the query by sending Alice "CAS signs auth(shaketable, Alice)", then Alice can push that fact to Bob and repeat her earlier query. (If CAS does not give Alice a suitable release policy for her to push that fact to Bob, she can query CAS for the policy she needs: ?CAS signs srelease(CAS signs auth(shaketable, Alice), Alice, Bob).) This time, Bob can use the instantiation, modus ponens, and signature derivation rules to prove "Bob signs auth(shaketable, Alice)". Bob is configured to send this signed fact to Alice, after proving that it is releasable (instantiation and modus ponens derivation rules). If he is also configured to send her all associated release policies, then she will be able to send the authorization fact to anyone. If he does not automatically send her the release policy, she and her proxies will have to query him for release permission each time they send out the authorization fact.

8. CONCLUSIONS

We have presented a brief overview of the PeerAccess framework, concentrating on its handling of base and release policies, and shown how it can be used in reasoning about the behavior of resource owners, their clients, and the Community Authorization Service deployed on supercomputing grids. We have also presented

a formal semantics and proof theory for PeerAccess, and shown their equivalence in the Appendix.

The features of PeerAccess were motivated by our need to model certain run-time authorization activities supported in the Grid Security Infrastructure. To meet these needs, PeerAccess allows one to model the local reasoning of individual peers who are unaware of the internal state of other peers. PeerAccess also allows one to reason about possible future global evolution of the system (e.g., for safety or liveness analysis). PeerAccess supports peer autonomy in choice of run-time behavior; this behavior is encoded in individual peers' ECA rules, exposure policies, and proof hints, and expressed in a peer's choice of pushing or pulling information, its willingness to accept pushed information and queries, and how hard it will work to answer the queries it accepts (i.e., what other peers it is willing to contact for help). Peers can easily describe their purpose in asking a query, and the answering peer can easily limit the purposes for which the answers will be used (subject to voluntary compliance). PeerAccess offers an extensible set of features, including the ability to model a variety of kinds of information release policies (including the sticky release policies used in the CAS examples); non-repudiable, verifiable communications between peers; easy ways to limit a peer's effort to prove a conclusion, and to direct its efforts in the most promising directions, through the use of proof hints; modeling of the interface a peer exposes to the outside world, through exposure policies; and potential easy extension of the underlying language for particular scenarios, such as constraint Datalog, simple forms of negation, or additional types of policies, such as audit policies. Total freedom in peer behavior can lead to total chaos in run-time results, and PeerAccess offers an excellent base for modeling, comparing, and experimenting with different proposals for controlling peer run-time behavior through multi-party trust negotiation strategies and credential discovery algorithms.

9. ACKNOWLEDGMENTS

Winslett's research was supported by NSF under grants CCR-0325951 and IIS-0331707 and by an NCSA Fellowship. Bonatti's research was partially supported by the EU FP6 Network of Excellence REWERSE (IST-2004-506779). Zhang is also associated with Cisco Systems Inc., USA. We thank W. Nejdl and D. Olmedilla for discussions leading to the creation of PeerAccess.

10. REFERENCES

[1] J. Basney, W. Nejdl, D. Olmedilla, V. Welch, and M. Winslett. Negotiating trust on the Grid. In *2nd Workshop on Semantics in P2P and Grid Computing*, New York, 2004.

[2] L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, Berkeley, May 2005.

[3] M. Y. Becker and P. Sewell. Cassandra: distributed access control policies with tunable expressiveness. In *5th IEEE International Workshop on Policies for Distributed Systems and Networks*, Yorktown Heights, June 2004.

[4] M. Y. Becker and P. Sewell. Cassandra: flexible trust management, applied to electronic health records. In *IEEE Computer Security Foundations Workshop*, 2004.

[5] M. Blaze, J. Feigenbaum, and A. D. Keromytis. KeyNote: Trust management for public-key infrastructures (position paper). *Lect. Notes in Computer Science*, 1550:59–63, 1999.

[6] P. Bonatti and P. Samarati. Regulating Service Access and Information Release on the Web. In *Conference on Computer and Communications Security*, Athens, Nov. 2000.

[7] P. A. Bonatti and D. Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *Workshop on Policies for Distributed Systems and Networks*, 2005.

[8] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Trans. on Comp. Systems*, 8(1), 1990.

[9] J. Camenisch and E. V. Herreweghen. Design and implementation of the *idemix* anonymous credential system. In *Computer and Communications Security*, 2002.

[10] R. Gavriloaie, W. Nejdl, D. Olmedilla, K. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *European Semantic Web Symposium*, 2004.

[11] A. Herzberg, Y. Mass, J. Michaeli, D. Naor, and Y. Ravid. Access control meets public key infrastructure, or: assigning roles to strangers. In *Symp. on Security and Privacy*, 2000.

[12] A. Hess, J. Jacobson, H. Mills, R. Wamsley, K. E. Seamons, and B. Smith. Advanced client/server authentication in TLS. In *Network and Dist. Systems Security Symp.*, 2002.

[13] T. Jim. SD3: A trust management system with certified evaluation. In *IEEE Symp. on Security and Privacy*, 2001.

[14] T. Jim and D. Suciu. Dynamically distributed query evaluation. In *Principles of Database Systems*, 2001.

[15] G. Karjoth, M. Schunter, and M. Waidner. Platform for enterprise privacy practices: Privacy-enabled management of customer data. In *Workshop on Privacy Enhancing Technologies*, 2002.

[16] H. Koshutanski and F. Massacci. Interactive trust management and negotiation scheme. In *Workshop on Formal Aspects in Security and Trust*, Aug. 2004.

[17] N. Li and J. Mitchell. RT: A role-based trust-management framework. In *Third DARPA Information Survivability Conference and Exposition*, Apr. 2003.

[18] N. Li, W. Winsborough, and J. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1), Feb. 2003.

[19] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and C. Tuecke. A community authorization service for group collaboration. In *Workshop on Policies for Distributed Systems and Networks*, 2002.

[20] B. Pfitzmann and M. Waidner. Federated identity-management protocols–where user authentication protocols may go. In *11th Cambridge International Workshop on Security Protocols*, Apr. 2003.

[21] C. Ruan, V. Varadharajan, and Y. Zhang. Logic-based reasoning on delegatable authorizations. In *Foundations of Intelligent Systems, ISMIS 2002*, Lyon, June 2002.

[22] S. Staab, B. Bhargava, L. Lilien, A. Rosenthal, M. Winslett, M. Sloman, T. S. Dillon, E. Chang, F. K. Hussain, W. Nejdl, D. Olmedilla, and V. Kashyap. The pudding of trust. *IEEE Intelligent Systems*, 19(5):74–88, Sep./Oct. 2004.

[23] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *Workshop on Formal Methods in Security Engineering*, Oct. 2004.

[24] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu. The TrustBuilder architecture for trust negotiation. *IEEE Internet Computing*, 6(6), 2002.

[25] T. Yu, M. Winslett, and K. E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. on Info. and System Security*, 6(1), 2003.